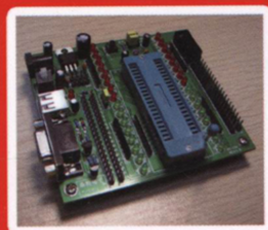


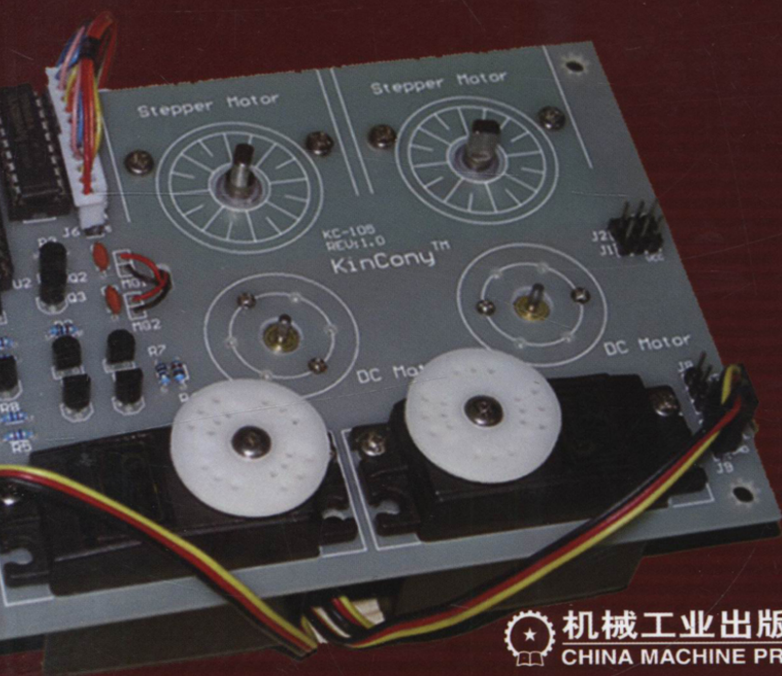
C51 DANPIANJI GAOXIAO RUMEN



C51 单片机高效入门

第2版

徐玮　等编著





视频演示光盘





机械工业出版社
CHINA MACHINE PRESS

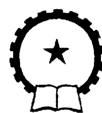
VISIT…

LANZAROTE Caliente.com

C51 单片机高效入门

第 2 版

徐玮　等编著



机 械 工 业 出 版 社

本书是以目前最为流行的8051系列单片机为主体，同时使用C程序设计语言来进行描述的。全书共分为四部分内容：单片机基础知识、C语言程序设计、单片机入门基础实例、单片机高级应用实例。以理论与实践相结合的方式来进行讲解，避免了传统教科书给人枯燥、乏味的感觉。讲解风格通俗易懂、条理清晰、实例丰富、图文并茂，即使是没有任何单片机基础的人，也可以通过本书的学习，踏入单片机世界的大门。

作者为本书的出版开发了相应的学习编程、仿真及实验板，以方便读者朋友进行学习，同时以大量实例照片记录了实验的过程及现象，以激发读者朋友对单片机的兴趣爱好。

本书的配套光盘包含了所有实验的源程序代码、一些常用的电子工具软件、芯片资料、实验过程照片以及实验演示视频录像。因此，通过本书，读者获得的是教程和学习平台的结合，不仅可以用于学习，而且还可以用于工厂、企业的产品研发。

本书可供电子爱好者和大学、中专相关专业学生参考。

图书在版编目（CIP）数据

C51单片机高效入门/徐玮等编著.—2版.—北京：机械工业出版社，2010.4（2011.2重印）

ISBN 978-7-111-30335-0

Ⅰ.①C… Ⅱ.①徐… Ⅲ.①单片微型计算机 Ⅳ.①TP368.1

中国版本图书馆CIP数据核字（2010）第063935号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
责任编辑：林春泉 责任校对：李秋荣
封面设计：路恩中 责任印制：乔 宇
北京机工印刷厂印刷（三河市南杨庄国丰装订厂装订）
2011年2月第2版第2次印刷
184mm×260mm·25.5印张·627千字
3 001—5 000册
标准书号：ISBN 978-7-111-30335-0
ISBN 978-7-89451-488-2（光盘）
定价：55.00元（含1CD）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：（010）88361066	门户网：http：//www.cmpbook.com
销 售 一 部：（010）68326294	教材网：http：//www.cmpedu.com
销 售 二 部：（010）88379649	
读者服务部：（010）68993821	**封面无防伪标均为盗版**

前　　言

当今世界科学技术飞速发展，以前您需要花费大量的时间和精力来搭建一个模拟电路，繁多的元器件增加了产品的成本；而现在，只需要一块几平方厘米大小的单片机，再写入相应功能的程序，便可以代替以前的老电路了。相信您在使用并掌握了单片机技术后，无论在今后开发或是工作上，都会带来意想不到的惊喜。

本书的编著者着眼于“高效入门”、“趣味学习”、“学以致用”的指导思想。全书以理论与实践相结合为主线，能够使读者轻松快捷地掌握单片机基础知识，并使读者朋友具有初步开发设计单片机产品的能力。本书讲解风格通俗易懂、条理清晰、实例丰富、图文并茂，即使您是一位单片机的门外汉，相信您看了本书以后，也能运用单片机的知识来解决一些实际问题，将知识转化为生产力。

全书共分为四部分内容：单片机基础知识、C 语言程序设计、单片机入门基础实例、单片机高级应用实例。

单片机基础知识：介绍单片机的发展历史，揭开它的神秘之处。告知读者所关心的一个实际问题：单片机到底能够做哪些应用，这也是我们为什么要学习单片机技术的原因。当我们明确了学习的目标后，肯定需要做好学习实践平台的准备，在此，我们将一一为读者进行讲解单片机学习的有效方法与途径。其次，将为读者陆续讲解单片机的内部结构、引脚定义、存储器、寄存器、定时/计数器、中断系统、串行通信等相关知识，让读者对单片机有实质性的了解。

C 语言程序设计：经常会有人问，应用单片机技术是用 C 语言好，还是用汇编语言好，这两种语言有各自的特点。汇编语言的优点是比较灵活，但程序不易理解，对产品的移植、升级不太有利；而 C 语言已有了非常丰富的库函数供用户使用，因为它是高级语言，程序代码的编写也非常人性化，易于阅读、理解，C 语言已经成为了一门在整个计算机领域普遍应用的语言了。因此，本书也是以 C 语言来进行描述的，我们将会向读者介绍 C 语言的数据类型、运算符、表达式，分支与循环控制语句，编译预处理与位运算，数组与函数，指针、结构体与共用体等知识，使读者具有 C 语言程序设计的能力。

单片机入门基础实例：由于单片机是一门实践性非常强的学科，即使您有再多的理论基础，也必须通过较多的实际操作才能真正学好这门技术。因此，在这部分章节中，我们将为读者朋友先引入一系列具有趣味性、简单易懂的基础实验实例，如点亮一个发光管，流水灯控制，按键、蜂鸣器、数码管、继电器的操作和使用，串行通信等。在此，我们暂时不求技术上的深入，只求让读者明白单片机到底是如何实现我们所需要的特定功能的，我们又是如何通过软件的程序，最终从硬件功能上反映出来的。

单片机高级应用实例：熟悉了基础实例，想必读者朋友已经对单片机有了一定程度的认识，知道自己实现什么样的功能，应该写什么样的程序。在这部分内容中，我们将为读者朋友做一些单片机高级应用实例的介绍，让您从学习单片机知识的水平提升到产品开发的高度。实例有液晶显示、步进电动机控制、I^2C 总线原理、数字温度传感器应用、无线通信控制、多功能器件 X25045/X5045 的应用、红外线遥控的软件解码、模/数转换器应用实例、DS1302 时钟芯片的应用等。看完这部分内容，相信您已经跨入了单片机世界的大门，并具有初步的产品开发能力了，接下来便是靠时间来积累实践经验了，相信只要发挥您的想象，一定可以将单片机发挥出更大的潜力。

为了方便广大读者朋友的学习交流，读者朋友可以访问我们的网站。同时，如果您对本书中所用到的学习器材、设备有兴趣的话，也可以访问我们的网站查看购买方法。当然，更新更详细的学习资料及内容，我们也都会定期放到网上供大家使用。

网址：http：//www. hificat. com

最后，特别感谢各位同事和朋友的热心帮助，使得本书能够顺利完成，参与撰写的还有：庄建清、杨丹枫、彭敏芳、魏巍、邵晶晶、戴婧、徐金林、卢水英、卢剑、邵磊、韩珈骏、蔡东琦、徐富军、许敏、孙燕、沈媛媛、金向红，也特别感谢机械工业出版社林春泉编辑为我们提供的帮助。我们衷心盼望本书能够对从事单片机技术工作的朋友有所帮助。

由于本书程序实例和演示图表都比较多，作者水平有限，难免会有错误与不妥之处，不足之处请广大读者批评指正。

编著者

2010 年 2 月

目　　录

前言
第 1 章　初识单片机 …… 1
1.1　单片机及其发展历史 …… 1
1.2　单片机到底能够做哪些应用 …… 2
1.3　学习单片机软、硬件实验设备的准备 … 7
1.4　单片机学习的有效方法与途径 …… 13
第 2 章　单片机基础知识 …… 15
2.1　MCS-51 单片机内部结构 …… 15
2.1.1　MCS-51 单片机组成框图 …… 15
2.1.2　MCS-51 单片机工作机制 …… 16
2.1.3　MCS-51 单片机内部功能部件 …… 17
2.2　引脚定义与特性 …… 18
2.3　MCS-51 单片机存储器和寄存器 …… 19
2.3.1　MCS-51 单片机的存储器结构 …… 19
2.3.2　MCS-51 单片机的寄存器 …… 20
2.4　定时/计数器 …… 22
2.4.1　定时/计数器概述 …… 22
2.4.2　定时/计数器结构 …… 22
2.4.3　定时/计数器控制寄存器 …… 23
2.4.4　定时/计数器的工作方式 …… 24
2.4.5　定时/计数器的应用 …… 26
2.4.6　定时器的应用 …… 27
2.5　MCS-51 单片机中断系统 …… 28
2.5.1　单片机中断 …… 28
2.5.2　中断的必要性 …… 29
2.5.3　中断源 …… 29
2.5.4　中断优先级 …… 29
2.5.5　中断响应过程 …… 29
2.6　中断系统 …… 30
2.6.1　中断系统结构 …… 30
2.6.2　MCS-51 中断源 …… 31
2.6.3　中断控制 …… 31
2.6.4　中断响应等待时间 …… 33
2.6.5　中断撤消 …… 33
2.6.6　中断系统应用举例 …… 33
2.7　串行通信 …… 35
2.7.1　串行通信概述 …… 35
2.7.2　MCS-51 单片机的串行接口结构 … 37
2.7.3　MCS-51 的串行口数据缓冲器 SBUF …… 37
2.7.4　串行通信控制寄存器 …… 37
2.7.5　波特率选择与设置 …… 40
2.7.6　RS232 标准接口总线及串行通信设计 …… 41
第 3 章　C 语言数据类型、运算符、表达式 …… 46
3.1　C 语言概论 …… 46
3.1.1　C 语言的发展过程 …… 46
3.1.2　C 语言的特点 …… 46
3.1.3　C 源程序的结构特点 …… 46
3.1.4　C 语言的字符集 …… 47
3.1.5　C 语言词汇 …… 48
3.2　数据类型、运算符与表达式 …… 49
3.2.1　C 语言的数据类型 …… 49
3.2.2　算术运算符和算术表达式 …… 61
3.2.3　关系运算符和表达式 …… 65
3.2.4　逻辑运算符和表达式 …… 67
第 4 章　分支与循环控制 …… 71
4.1　if 语句 …… 71
4.1.1　if 语句的 3 种形式 …… 71
4.1.2　if 语句的嵌套 …… 75
4.2　条件运算符和条件表达式 …… 77
4.3　switch 语句 …… 79
4.4　循环控制 …… 82
4.4.1　概述 …… 82
4.4.2　goto 语句以及用 goto 语句构成循环 …… 82
4.4.3　while 语句 …… 83
4.4.4　do-while 语句 …… 86
4.4.5　for 语句 …… 88
4.4.6　循环的嵌套 …… 90
4.4.7　break 和 continue 语句 …… 91
第 5 章　编译预处理与位运算预处理命令 …… 95

5.1　概述 …… 95
5.2　宏定义 …… 95
5.2.1　不带参数的宏定义 …… 95
5.2.2　带参数的宏定义 …… 97
5.3　文件包含 …… 99
5.4　条件编译 …… 100
5.5　位操作运算符 …… 102
第6章　数组与函数 …… 105
6.1　一维数组的定义和引用 …… 105
6.1.1　一维数组的定义方式 …… 105
6.1.2　一维数组元素的引用 …… 106
6.1.3　一维数组的初始化 …… 108
6.1.4　一维数组程序举例 …… 109
6.2　二维数组的定义和引用 …… 110
6.2.1　二维数组的定义 …… 110
6.2.2　二维数组元素的引用 …… 111
6.2.3　二维数组的初始化 …… 112
6.3　字符数组 …… 113
6.3.1　字符数组的定义 …… 113
6.3.2　字符数组的初始化 …… 113
6.3.3　字符数组的引用 …… 114
6.3.4　字符串和字符串结束标志 …… 114
6.4　函数概述 …… 114
6.4.1　函数定义的一般形式 …… 115
6.4.2　函数的参数和函数的值 …… 116
6.4.3　函数的返回值 …… 117
6.4.4　函数的调用 …… 117
6.4.5　被调用函数的声明和函数原型 …… 118
6.4.6　函数的嵌套调用 …… 119
6.4.7　函数的递归调用 …… 120
6.4.8　数组作为函数参数 …… 121
6.5　局部变量和全局变量 …… 123
6.5.1　局部变量 …… 123
6.5.2　全局变量 …… 125
第7章　指针、结构体与共用体 …… 127
7.1　指针和地址 …… 127
7.2　指针变量和指针运算符 …… 127
7.3　指针与函数参数 …… 131
7.4　指针、数组和字符串指针 …… 132
7.5　指针数组 …… 136
7.6　多级指针 …… 138
7.7　返回指针的函数 …… 139
7.8　函数指针 …… 140
7.9　结构与联合 …… 140
7.9.1　结构的定义 …… 140
7.9.2　结构数组 …… 143
7.9.3　结构与函数 …… 144
7.9.4　结构的初始化 …… 145
7.9.5　联合 …… 146
第8章　51单片机实验器材快速操作入门 …… 148
8.1　增强型51单片机实验板操作入门 …… 148
8.2　增强型51单片机实验板仿真操作指南 …… 150
8.3　增强型51单片机实验板仿真实例 …… 151
8.4　芯片烧写操作指南 …… 156
8.5　增强型51单片机实验板常见问题解答 …… 159
第9章　单片机入门基础实例 …… 161
9.1　点亮一个发光二极管 …… 161
9.1.1　实现方法 …… 161
9.1.2　源程序 …… 162
9.1.3　代码分析 …… 162
9.2　使发光二极管闪动 …… 163
9.2.1　实现方法 …… 163
9.2.2　源程序 …… 163
9.2.3　代码分析 …… 163
9.2.4　深入了解 …… 164
9.3　流水灯 …… 164
9.3.1　实现方法 …… 165
9.3.2　源程序 …… 166
9.3.3　代码分析 …… 167
9.3.4　深入了解 …… 167
9.4　按键操作 …… 168
9.4.1　实现方法 …… 168
9.4.2　源程序 …… 170
9.4.3　代码分析 …… 170
9.4.4　深入了解 …… 170
9.5　蜂鸣器的使用 …… 171
9.5.1　实现方法 …… 172
9.5.2　源程序 …… 172
9.5.3　代码分析 …… 173
9.6　数码管的使用 …… 173
9.6.1　实现方法 …… 174
9.6.2　源程序 …… 175

9.6.3 代码分析 …… 176
9.6.4 深入了解 …… 176
9.7 单片机继电器控制 …… 178
9.7.1 继电器的工作原理与分类 …… 178
9.7.2 继电器的控制电路 …… 179
9.7.3 单片机控制继电器 …… 179
9.8 单片机综合应用程序 …… 180
9.9 单片机串行口数据接收 …… 187
第10章 单片机高级应用实例 …… 192
10.1 矩阵键盘应用实例 …… 192
10.1.1 矩阵键盘简介 …… 192
10.1.2 矩阵键盘的工作原理 …… 192
10.1.3 矩阵键盘软硬件设计实例 …… 193
10.2 字符型LCD应用实例 …… 199
10.2.1 液晶显示概述 …… 199
10.2.2 1602字符型LCD简介 …… 200
10.3 步进电动机应用实例 …… 210
10.3.1 步进电动机概述 …… 211
10.3.2 步进电动机的基本参数 …… 213
10.3.3 步进电动机的驱动 …… 214
10.4 I^2C总线器件应用实例 …… 219
10.4.1 I^2C总线基本概念 …… 219
10.4.2 I^2C总线的系统结构 …… 219
10.4.3 I^2C总线接口 …… 220
10.4.4 I^2C总线的时钟信号 …… 220
10.4.5 I^2C总线的传输协议与数据传送 …… 221
10.4.6 I^2C总线接口器件应用 …… 222
10.5 93CXX系列存储器应用实例 …… 230
10.5.1 SPI总线简介 …… 230
10.5.2 93C46存储器的软硬件设计实例 …… 233
10.6 DS18B20数字温度传感器应用实例 …… 241
10.6.1 单总线（1-WIRE）技术介绍 …… 241
10.6.2 DS18B20简介 …… 242
10.6.3 DS18B20新性能 …… 243
10.6.4 DS18B20外形及引脚说明 …… 243
10.6.5 DS18B20内特性 …… 243
10.6.6 DS18B20温度测试软、硬件设计 …… 247
10.7 无线通信模块应用 …… 253
10.7.1 PT2262/PT2272编码/解码芯片原理简介 …… 254
10.7.2 编码发射模块简介 …… 256
10.7.3 解码接收模块 …… 256
10.7.4 PT2262/PT2272芯片的地址编码设定 …… 257
10.7.5 基于单片机的无线收发模块应用 …… 257
10.8 X25045/X5045多功能器件的应用 …… 261
10.8.1 看门狗、电压监控概述 …… 261
10.8.2 X25045/X5045的结构及工作原理 …… 262
10.8.3 X25045/X5045和单片机之间的软件接口程序设计 …… 264
10.9 红外遥控的软件解码 …… 267
10.9.1 红外遥控概述 …… 267
10.9.2 红外遥控的编码和软件解码方法 …… 271
10.9.3 遥控器软件解码的程序实现 …… 275
10.10 模/数转换器应用实例 …… 282
10.10.1 模/数转换器简介 …… 282
10.10.2 A/D转换器的主要技术指标 …… 284
10.10.3 串行A/D转换器ADC0832简介 …… 284
10.10.4 ADC0832应用实例 …… 286
10.11 DS1302的应用 …… 291
10.11.1 实时时钟芯片概述 …… 291
10.11.2 DS1302的结构及工作原理 …… 292
10.11.3 DS1302和单片机之间的接口程序实现 …… 294
10.12 12864点阵型LCD应用实例 …… 297
10.12.1 点阵型LCD的显示原理 …… 297
10.12.2 12864点阵型LCD简介 …… 298
10.12.3 12864点阵型LCD软硬件设计实例 …… 304
第11章 新型单片机外扩展模块 …… 317
11.1 KC-101 51/AVR单片机最小系统核心板 …… 317
11.2 KC-102单片机显示板模块 …… 321
11.3 KC-103单片机键盘板模块 …… 330
11.4 KC-104模数/数模转换模块 …… 339
11.5 KC-105电动机驱动模块 …… 348
11.6 KC-106单片机总线模块 …… 352
11.7 KC-201 FM立体声收音模块 …… 363
11.8 KC-202电视信号接收模块 …… 379
附录 Keil开发软件的介绍 …… 385
参考文献 …… 397

第 1 章　初识单片机

科技的进步需要技术不断的提升。一个大而复杂的模拟电路花费了你巨大的精力，繁多的元器件增加了你的成本。而现在，只需要一块几厘米见方的单片机，写入简单的程序，就可以使以前的电路变得简单的多。相信你在使用并掌握了单片机技术后，今后不管你在开发或是工作上，一定会带有意想不到的惊喜。

1.1　单片机及其发展历史

相信大家对个人计算机（PC——Personal Computer）已不再陌生，计算机已进入千家万户，一台完整的计算机系统有以下部分构成：CPU——Centra Processing Unit（中央处理器）、RAM——Random Access Memory（随机存取存储器）、ROM——Read Only Memory（只读存储器）、输入/输出设备（如串行口、并行口等）。在 PC 上这些部分由若干集成电路做成相应功能的板卡，如果你拆开计算机机箱，你就会看到一系列大大小小的板卡插在主板上。

我们通常所讲的“单片机”又称微控制器，它并不是完成某一个逻辑功能的芯片，而是将上述那些系统集成到一块集成电路芯片中去了。技术在进步，现在某些型号的单片机芯片中也集成了 A/D——Aralog to Digital Conversion（模拟、数字转换），D/A——Digital-to-Analog Conversion（数字、模拟转换）等功能模块。简单地讲：这块芯片就成了一台计算机，它具有体积小、重量轻、价格低廉等特点。

PC 的售价要几千元，甚至上万元，这不是个小数目，而单片机芯片集成了众多的功能模块后，价格却并没有像 PC 那样高，从几元至几十元不等。当然不同型号的单片机芯片体积也会有所不同，如有些是 20 引脚封装的，有些是 40 引脚封装的，这主要取决于它的功能。一般来说，引脚多的要比引脚少的功能强大。

可能你会问，为什么会这样呢？其实道理很简单，用户可以根据各自不同的用途来选择合适的芯片型号，打个比方：现在市场上的品牌计算机有这么多，为什么有的贵、有的便宜？如果你爱打游戏，那就选显卡好的；如果你用来做图形工作，那就选内存大的；如果你的数据资料多，那就选择大硬盘的计算机。总而言之，各取所需。

单片机的历史：

第一代：20 世纪 70 年代后期，4 位逻辑控制器件发展到 8 位。使用 NMOS——N-channel Metal-Oride Semiconductor（N 沟道金属氧化物半导体）工艺（速度低，功耗大、集成度低）。代表产品：MC6800、Intel 8048。

第二代：20 世纪 80 年代初，采用 CMOS——Complementary Metal-Oxide-Semiconductor（互补金属-氧化物-半导体）工艺，并逐渐被高速低功耗的 HMOS——High-speed Metal-Oxide-Semiconductor（高速金属氧化物半导体）工艺代替。代表产品：MC146805、Intel 8051。

第三代：近 10 年来，MCU 的发展出现了许多新特点：

1）在技术上，由可扩展总线型向纯单片型发展，即只能工作在单片方式。

2）MCU 的扩展方式，从并行总线型发展出各种串行总线。

3）将多个 CPU 集成到一个 MCU——Multi-Chip Unit（多芯片单元）中。

4）在降低功耗，提高可靠性方面，MCU 工作电压已降至 3.3V 。

第四代：FLASH 的使用，使 MCU 技术进入了第四代。

1.2 单片机到底能够做哪些应用

目前，单片机已渗透到我们工作、生活的各个领域，几乎很难找到哪个领域没有单片机的踪迹了。导弹的飞行装置靠的是单片机，网络数据通信及传输，工业自动化控制，智能 IC 卡系统及各类家用电器的控制都离不开单片机。单片机的特点是体积小，在其增加一些外围电路之后，就能成为一个完整的应用系统。例如，我们日常生活中所用的数字电子秤，其内部就有一块单片机芯片，再加上传感器、液晶屏和一些附加电路，就形成了一个完整的应用系统。由此可见，单片机的可扩展性是不错的，应用也相当灵活。

下面我们将由浅入深地讲解单片机应用实例，希望能给初学者朋友们带来一些感性的认识，让大家知道单片机到底能够干什么，都有哪些具体的应用。

1）应用实例之一：如图 1-1 所示，电路板左上角显示“8.8.8.8”数字字样，其显示部件叫“七段数码管”，它在家电及工业控制中有着很广泛的应用，例如用来显示温度、数量、重量、日期、时间等，具有显示醒目、直观的优点。如果你弄明白了数码管显示的基本原理知识，做一些电子钟、计数器之类的应用系统将不成问题。

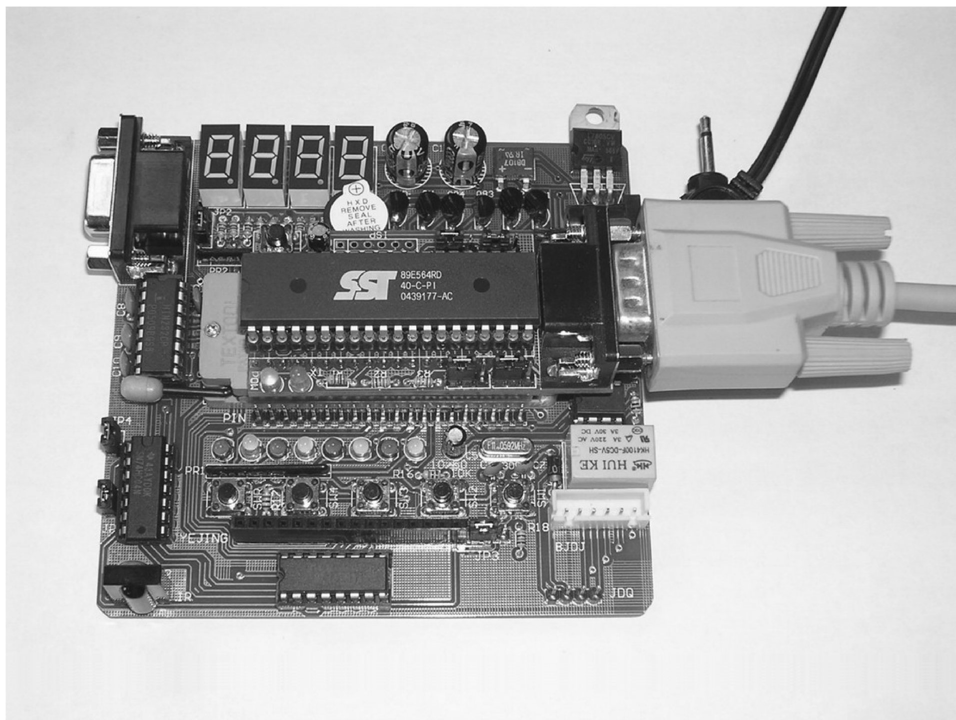


图 1-1　数码管显示实例

2）应用实例之二：如图 1-2 所示，这是一个液晶屏显示的应用实例。1602 型液晶屏是

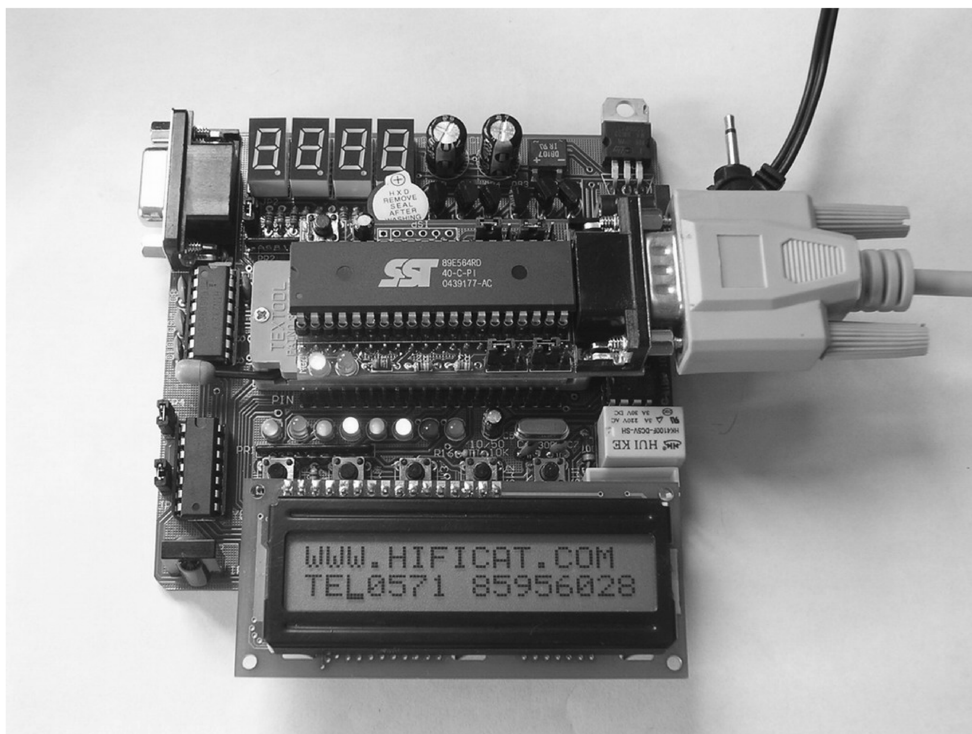


图 1-2　1602 液晶屏显示实例

一种用 5 ×7 点阵图形来显示字符的液晶显示器，根据显示的容量可以分为 1 行 16 个字、2 行 16 个字、2 行 20 个字等。常用的为 2 行 16 个字，它可以显示我们通过单片机在液晶屏上显示的网址和电话号码。液晶屏与数码管相比，显得更为专业，漂亮。液晶屏以其微功耗、体积小、显示内容丰富、超薄轻巧等诸多优点，在袖珍式仪表和低功耗应用系统中得到了越来越广泛的应用，如 IC 卡电话机、液晶电子表等各类显示设备。

3）应用实例之三：如图 1-3 所示，这是一个温度测试及控制系统的应用实例。液晶屏实时显示温度值，通过按键设定温度报警上、下限数值。当实际温度超过上、下限额定温度值时，继电器产生触发动作，蜂鸣器报警。人工设定的温度报警值自动存入 DS18B20 温度传感器的 EEPROM——Electrically Erasable（and）Programmable Read-Only Memory（电可擦可编程只读存储器）中，可永久保存。每次开机时，自动从温度传感器的 EEPROM 读出温度报警值，这样就不用重复设置额定值了。图 1-4 所示为 DS18B20 温度传感器的实物图片。

4）应用实例之四：如图 1-5 所示，我们通过软件改变步进电动机各相电压，使其转动，并通过设置相应的延时值来达到调速的目的，步进电动机与传统玩具电动机有所不同，它是一种将电脉冲转化为角位移的执行机构。通俗地讲：当步进驱动器接收到一个脉冲信号，它就驱动步进电动机按设定的方向转动一个固定的角度。你可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时你可以通过控制脉冲频率来控制电动机转动的速度和加速度，从而达到调速的目的。步进电动机技术的应用也非常广泛，如打印机喷头的移动、安防系统中视频摄像头的转动等都是通过它来控制的。

5）应用实例之五：如图 1-6 所示，我们利用单片机来做液晶显示，与图 1-2 所示不同

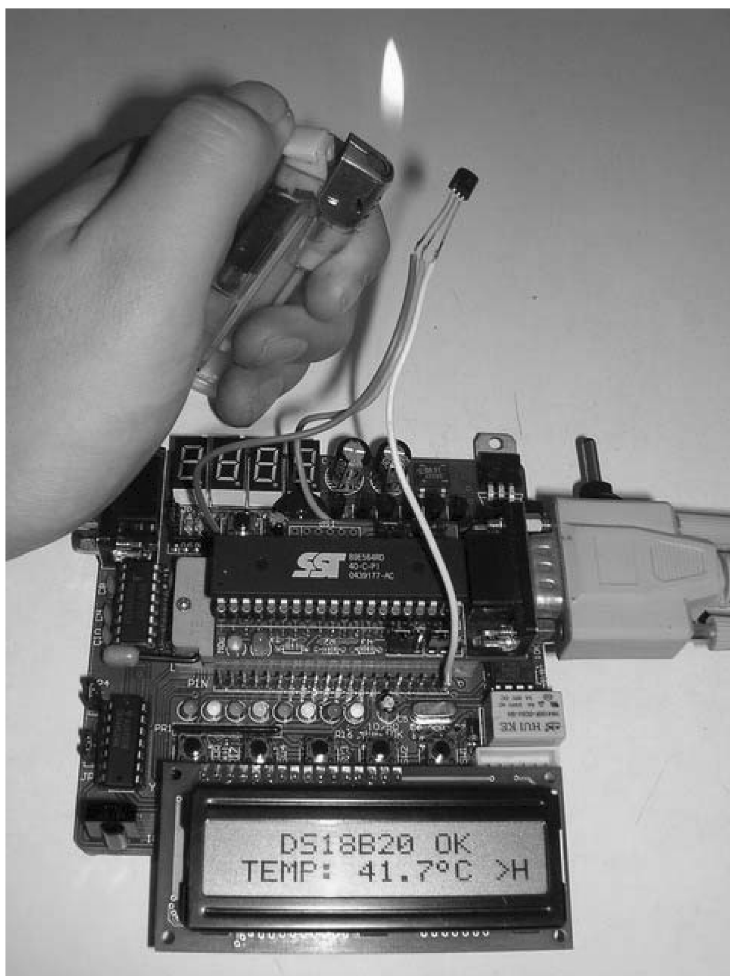


图 1-3　温度测试实例

之处在于我们使用的是 12864 的液晶屏，它可以显示各种字符及图形，可与 CPU 直接接口，具有 8 位标准数据总线、6 条控制线及电源线。采用 KS0107 控制 IC。通过取模软件，我们可以用来显示任何中文汉字及各种图形等。

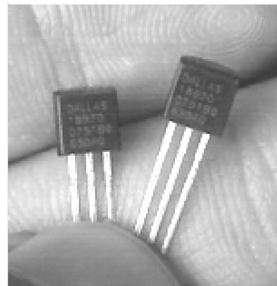


图 1-4　DS18B20 温度传感器

6）应用实例之六：红外线遥控是目前使用最广泛的一种通信和遥控手段。由于红外线遥控装置具有体积小、功耗低、功能强、成本低等特点，因而继彩电、录像机之后，在录音机、音响设备、空调机以及玩具等其他小型电器装置上也纷纷采用了红外线遥控。工业设备中，在高压、辐射、有毒气体、粉尘等环境下，采用红外线遥控不仅完全可靠，而且能有效地隔离电气干扰。如图 1-7 所示，通过按遥控器上的数字键“1～8”，实验板完成解码功能，并通过数码管显示其相应的键值。当然，你也可以改写程序，让红外线遥控器来控制实验板上的继电器，或者通过红外线遥控器让实验板上的蜂鸣器唱歌，这些并非难事，只要发挥你的想象，你可以想出各种控制方法。

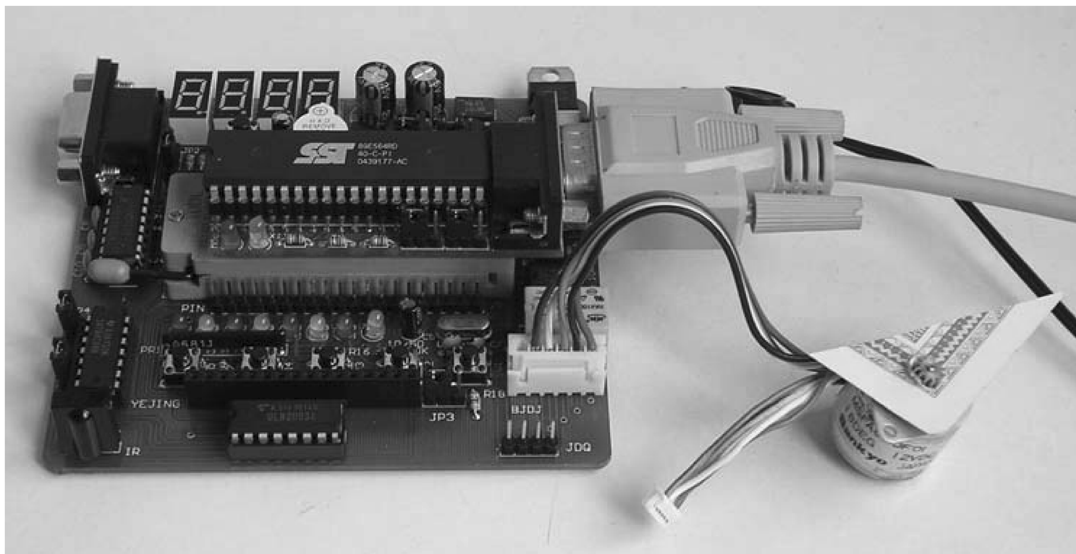


图 1-5　步进电动机控制实例

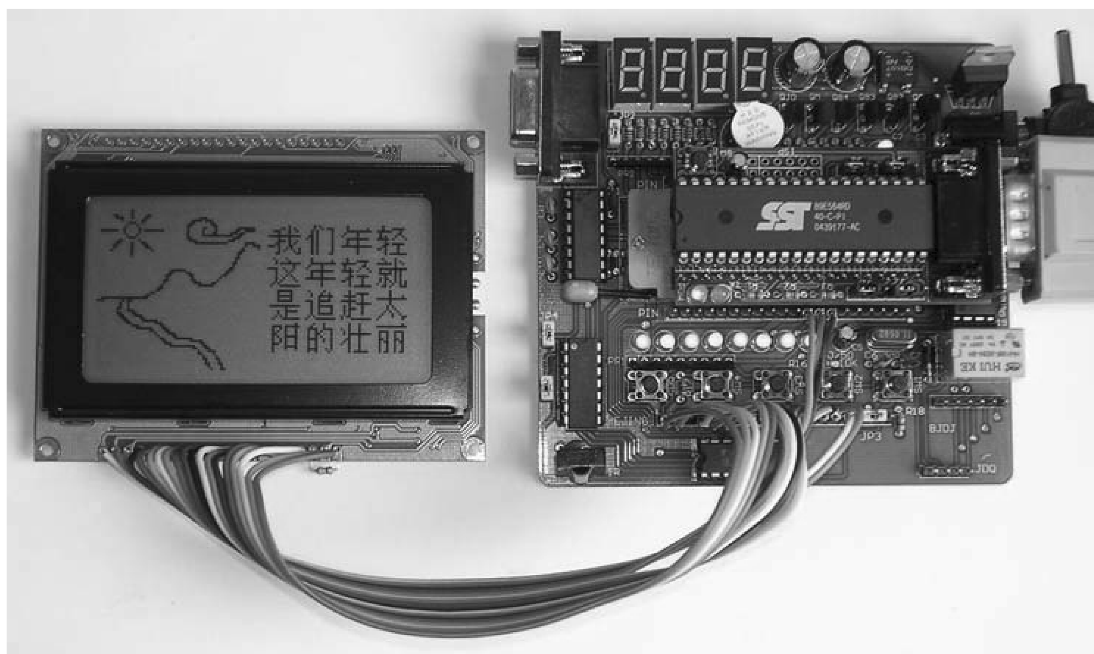


图 1-6　12864 液晶显示实例

7）应用实例之七：红外线遥控的缺点是有方向性，即遥控发射器需要对准遥控接收器才能起到控制作用。无线电遥控的方式就克服了这个缺点，它没有方向性，如图 1-8 所示的是 200m 的无线遥控器，电源、灯泡、继电器串联接入电路，然后通过手持式无线遥控器来控制，人体距离实验板的最大距离为 200m；如果换成 1000m 的发射器，就可以进行 1000m 的无线遥控了，通过这样的原理，可以进行各种无线遥控类的产品开发。

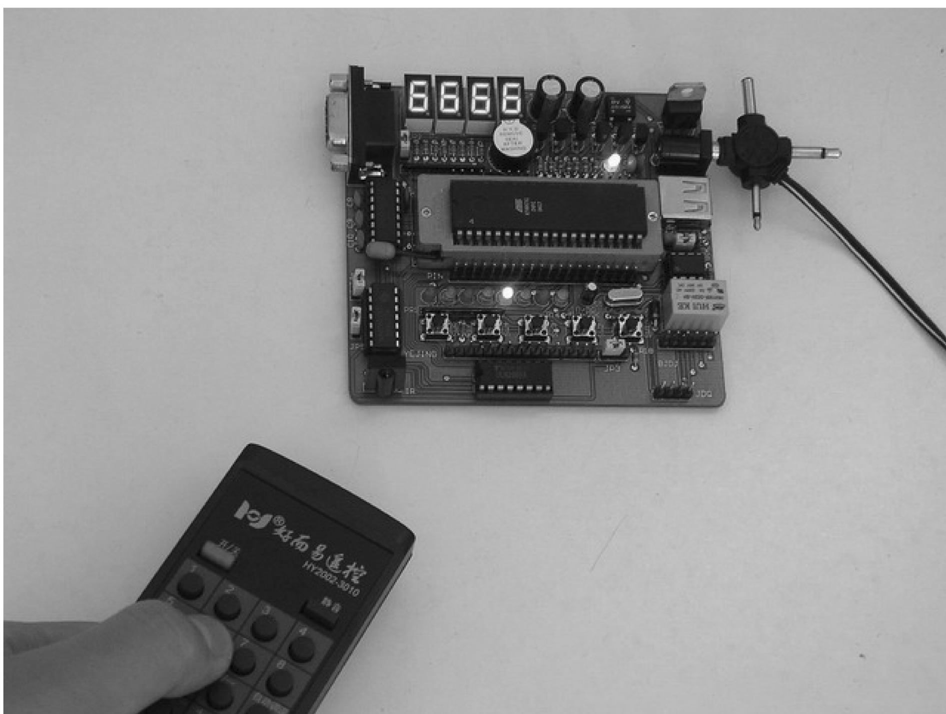


图 1-7　红外线遥控数码管显示实例

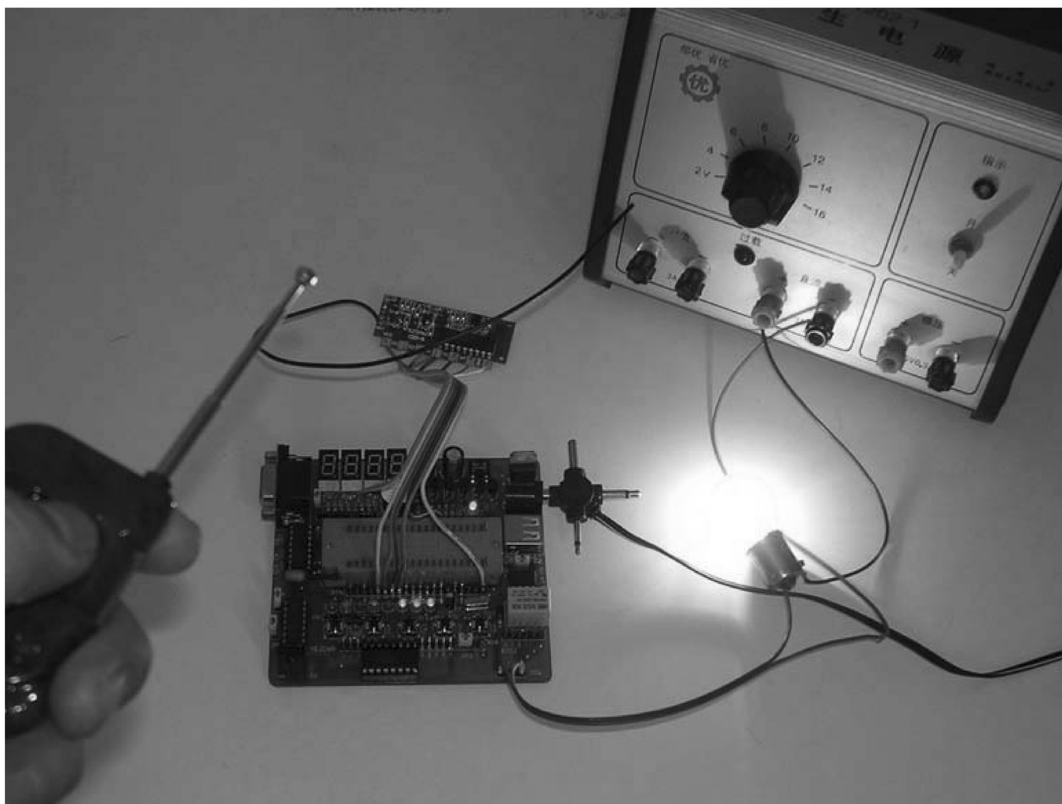


图 1-8　无线电遥控继电器开关灯泡实例

1.3 学习单片机软、硬件实验设备的准备

在你决定学习单片机之前，首先请你准备好必要的软、硬件设备，完善的学习条件才能给你带来高效的学习收获。

硬件准备：计算机一台（奔腾级以上的家用计算机即可，要求不苛刻）；烧写 51 单片机芯片的编程器一台；51 实验板一块（单片机实验的核心部分）；仿真器一套，它会给你的学习带来很大的方便，提高学习效率；实验附件 1602LCD 液晶屏、红外线遥控器、步进电动机、DS18B20 温度传感器、200m 无线收发模块。

1. 增强型 51 实验板

首先，我们来看实验板的硬件资源可做哪些实验及其主要特点，实验板的实物照片如图 1-9 所示。

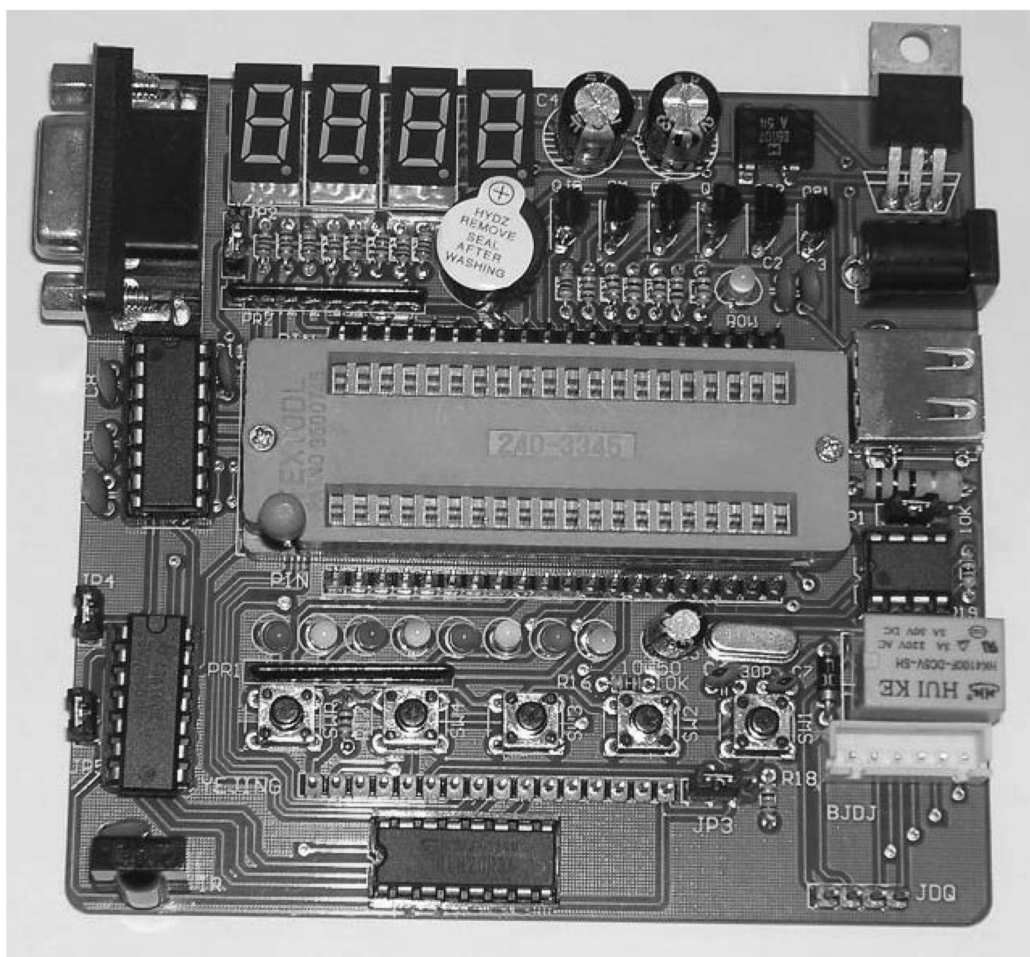


图 1-9　增强型 51 实验板

1）数码管：可以实验和仿真各种计数器、数字显示以及用单片机做电子钟等仿真。例如计数器、秒表、电子钟等。

2）LED 流水灯：可以显示 P 口的状态，与教程相配套，可做实验，如正反流水灯、交通指示灯等。

3）键盘：可以实验与键盘有关的程序。

4）扬声器：适合做各类发声程序的仿真和实验，如让小扬声器演奏各种乐曲，唱首歌。

5）继电器：有了它我们就可以知道怎么来做一个以弱控强——弱电控制强电的系统。

6）24C02：用来做 IIC 通信实验，当然你也可以更换不同芯片来做实验。

7）液晶屏：通过液晶屏显示你想要的信息，例如发光管、数码管显示更为漂亮，专业化。

8）RS232 串行接口：支持串口通信实验，可以让你的计算机和单片机互相通信，完成指定的任务。

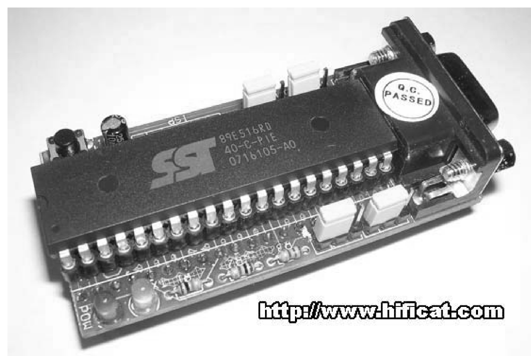
9）步进电动机驱动电路：可以非常方便地接上步进电动机，完成步进电动机的各类实验，如电动机的正、反转等。

10）红外线接收器：可以做红外线解码实验，红外线遥控器等，酷!!! 配合 SAA3010T 遥控器完成遥控解码及红外线遥控实验。如按遥控器的数字键 1 ~ 8，即可点亮实验板上的第一个发光管至第八个发光管，或按遥控器键数码管显示相应的数字。当然，你也可以通过改动程序来达到红外线遥控其他资源的目的。

11）所有芯片引脚都接有外扩排针，有利于外扩更多的功能，外扩实验的功能没有限制，完全由用户决定。

2. 微型 51 仿真器

现在我们对实验板已经有所了解，如图 1-9 所示，但仅有实验板还是无法完成我们的实验过程的，我们还需要通过仿真器与实验板相结合来调试程序，比如我们的程序有 50 行，假设代表了 5 个驱动硬件的动作，这时候如果有仿真器的话，我们可以让这 5 个动作一个个地去执行，同时能够观察到在执行这 5 个动作的过程中，单片机内部的各单元状态是怎么样的，也就是可以细致地分析一下整个程序在硬件中的具体工作过程。这样我们就可以了解程序中是否有问题存在，所以叫做仿真。使用仿真器就不必为了改程序而反复地烧写芯片，同时可以使用单步运行、指定端点停止等功能，调试极为方便。仿真器的实物如图 1-10 所示。在早些年前，因为 51 芯片的存储器是 EPROM 的，反复烧写的寿命非常有限，开发程序只能靠专业的、昂贵的专业仿真器来完成，排除了所有错误之后再将 HEX 或 BIN 文件一次写



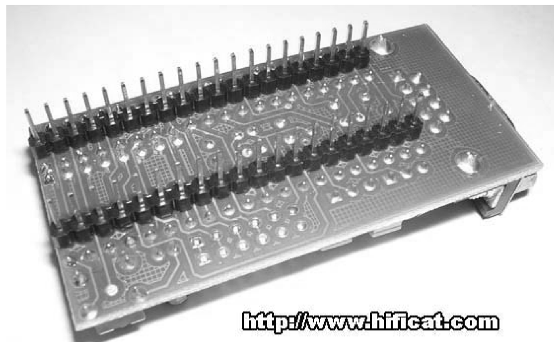


图 1-10　微型 51 仿真器

入单片机芯片内。现在，有了内部含有闪存的单片机之后，才使反复烧写试验成为可能，但是也还是无法实现像仿真器那样的实时调试，学习效率自然要低很多了。图 1-1 中，插在实验板上的那个设备就是仿真器。

仿真器可仿真 89C2051、89C51、89C52、89S51、89S52、89C58 等 MCS51 内核的单片机，直接支持 KEIL C51 的 IDE 开发仿真环境，RS232 通信接口，支持汇编、C 语言，混合调试。其兼容标准：仿真器具备的资源是 P0、P1、P2、P3 的 32 个 I/O 口，64K 程序空间，兼容 52 内核。

3. A51 编程器

当你使用仿真器和 51 实验板调试完程序后，最后一道工序就是将目标程序烧入芯片，我们通过使用编程器来完成这个步骤，通常也将编程器叫烧录器。图 1-11 所示为编程器烧写 51 单片机。

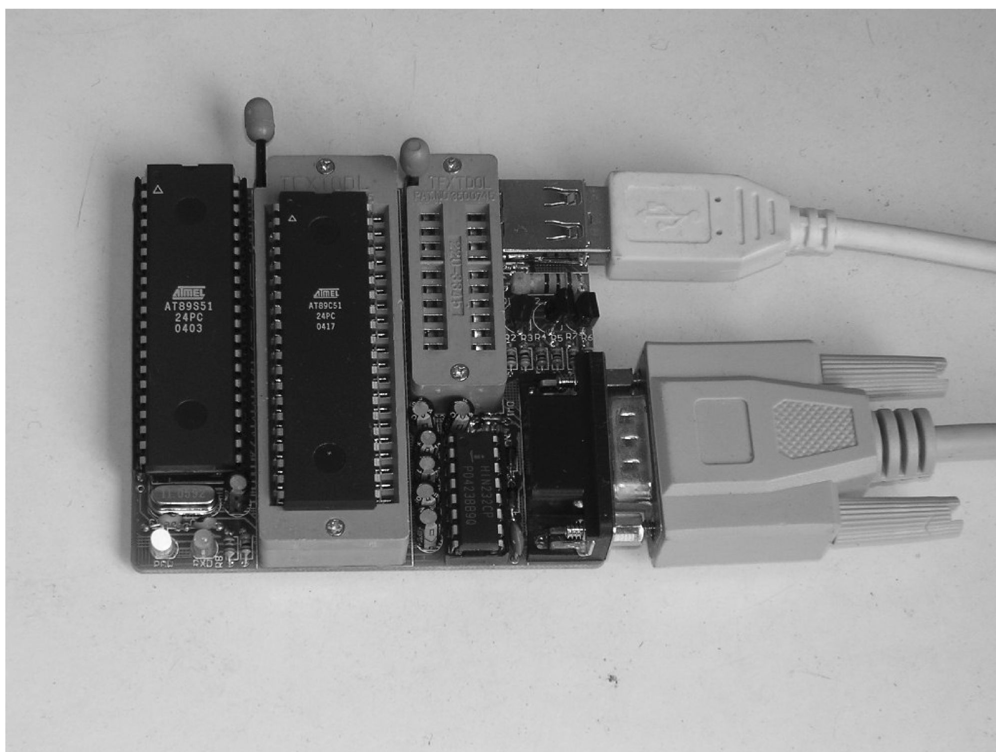


图 1-11　A51 编程器

A51 编程器支持目前最为经典和市场占有量最大的 ATMEL 公司生产的 AT89C51、C52、C55 和最新的 S51、S52；AT89C1051、2051、4051 等芯片，特别适合于渴望学习 51 单片机，又想尽量减小学习投入的朋友。毕竟 51 系列早已经成为了工业标准，学习 51 单片机，使一切都在单片机的控制下变得智能化，这是每一个爱好者和发烧友的梦想！

A51 编程器的主要特点：

1）使用串口通信，芯片自动判别，编程过程中的擦除、烧写、校验各种操作完全由编程器上的监控芯片 89C51 控制，不受 PC 配置及其主频的影响，因此烧写速度很快并且烧写速度和 PC 的档次无关。烧写成功率高达 100%。

2）采用 57600 高速波特率进行数据传送，编程速度可以和一般并行编程器相媲美。经测试，烧写一片 4K ROM 的 AT89C51 仅需要 9.5s，而读取和校验仅需要 3.5s。

3）体积小巧，省去笨重的外接电源适配器，直接使用 USB——Universal Serial Buy（通用串行总线）端口 5V 电源，携带方便，非常适合初学者学习 51 单片机的要求。

4）软件界面友好，菜单、工具栏、快捷键齐全，全中文操作，提供加密功能，可以保护你的创作产权。可以说是麻雀虽小，五脏俱全！

5）功能完善，具有编程、读取、校验、空检查、擦除、加密等系列功能。

6）40pin 和 20pin 锁紧插座，所有器件全部以第 1 引脚对齐，无附加跳线，对于 DIP 封装芯片无需任何适配器。

7）采用优质万用锁紧插座，和接触不良等问题彻底说再见，可烧写 40 引脚单片机芯片和 20 引脚单片机芯片。

8）改进的烧写深度确保每一片 51 系列芯片的反复烧写次数都能达到 1000 次以上！内部数据至少保存 10 年。

9）因为采用了 9 针串口通信，就不会再和打印机抢一个打印口，随时随地想烧就烧，让芯片编程成为一种快乐！

4. 实验附件介绍

实验附件——1602LCD 液晶屏、6121 码红外线遥控器、步进电动机、DS18B20 温度传感器、200m 无线收发模块。

（1）1602LCD 液晶屏

1602LCD 液晶屏正面如图 1-12 所示，反面如图 1-13 所示。

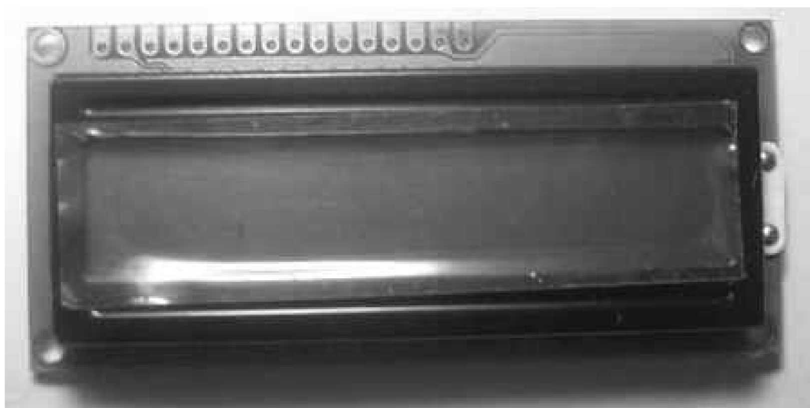


图 1-12　1602LCD 液晶屏正面

1602LCD 字符型液晶屏由 5×7 点阵字符位组成，可以用来显示数字、字母及各类符号，根据显示的容量为 2 行 16 个字，液晶屏带有绿色背光照明，显示效果清晰而美观。图 1-12 和图 1-13 为 1602LCD 液晶屏外观照片。

（2）红外线遥控器

红外线遥控器如图 1-14 所示。

（3）步进电动机

步进电动机是一种将电脉冲转化为角位移的执行机构。在没有超出负载的情况下，步进



图 1-13　1602LCD 液晶屏反面

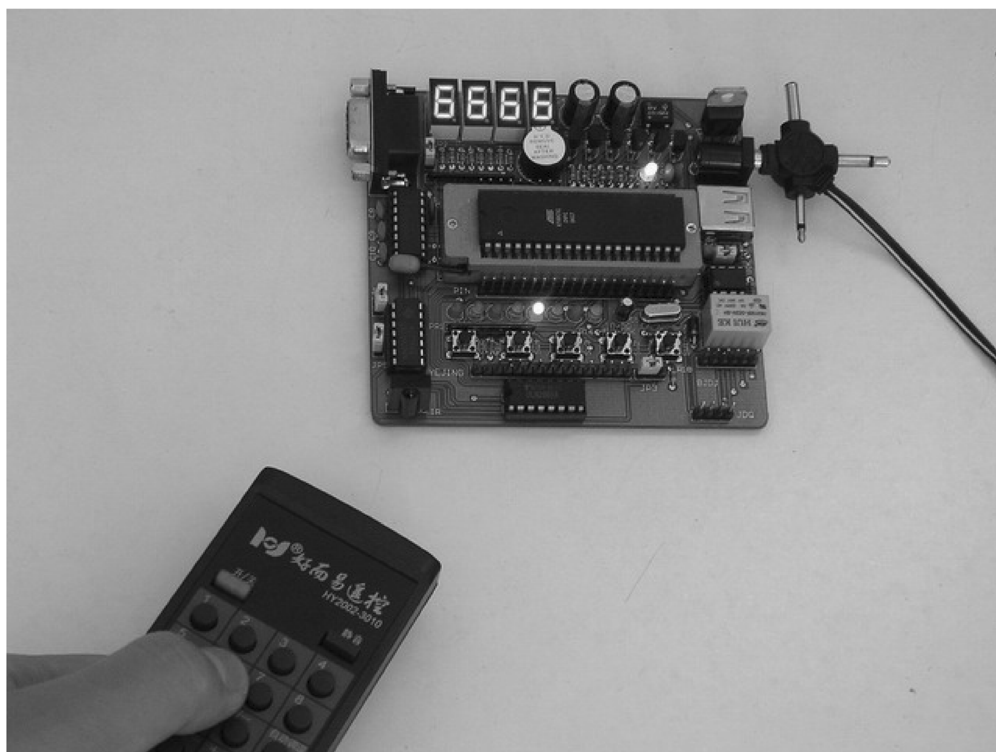


图 1-14　红外线遥控器

电动机的转动速度、停止的位置只取决于送给电动机脉冲信号的频率和脉冲数，而不会受到负载变化的影响，如：我们给步进电动机加一个脉冲信号，电动机则转过一个步距角。微型步进电动机如图 1-15 所示。

（4）DS18B20 温度传感器

DS18B20 是 DALLAS 公司生产的单总线式数字温度传感器，它具有微型化、低功耗、高性能、抗干扰能力强、易配处理器等优点，特别适用于构成多点温度测控系统，可直接将温度转化成串行数字信号（提供 9 位二进制数字）给单片机处理，且在同一总线上可以挂接多个传感器芯片。它具有 3 引脚 TO-92 小体积封装形式，温度测量范围为 −55℃ ~ +125℃，

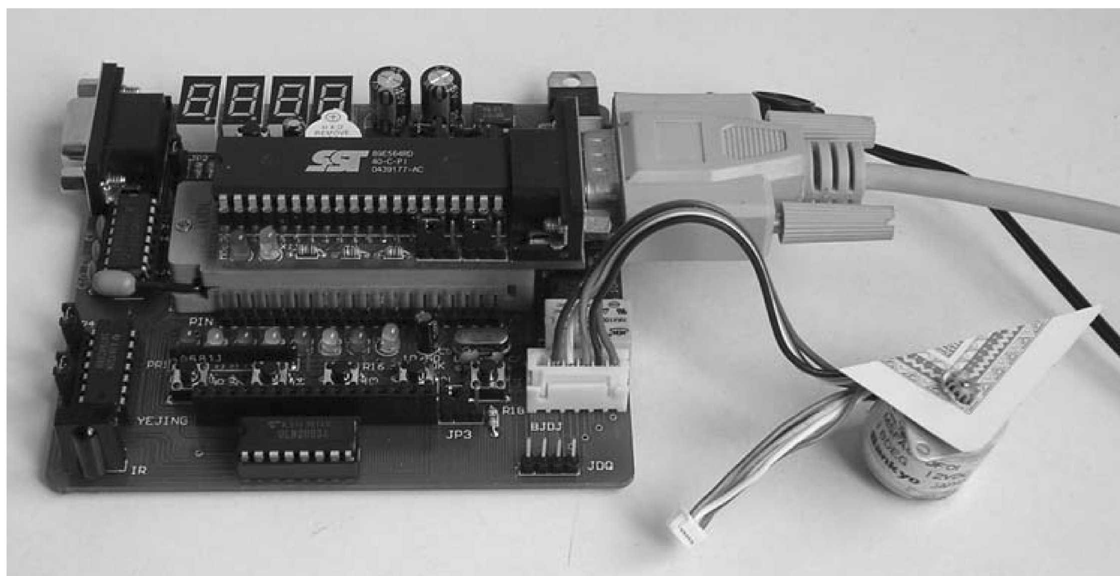
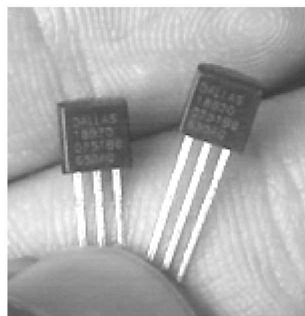


图 1-15　微型步进电动机

可编程为 9 位～12 位 A/D 转换精度，测温分辨率可达 0.0625℃，被测温度用符号扩展的 16 位数字量方式串行输出，其工作电源既可在远端引入，也可采用寄生电源方式产生，多个 DS18B20 可以并联到 3 根或 2 根线上，CPU 只需一根端口线就能与多个 DS18B20 通信，占用微处理器的端口较少，可节省大量的引线和逻辑电路。以上特点使 DS18B20 非常适用于远距离多点温度检测系统。如图 1-16 所示。

（5）200m 无线收发模块

200m 4 键遥控模块，常用于报警器设防、车库门遥控、摩托车、汽车的防盗报警等。遥控模块价格低廉，发射机手柄体积小巧、外观精致，耗电少，工作稳定可靠。采用优质塑料外壳，带保险盖，防止误碰按键。天线拉出时长 13cm，遥控器只有 20g 重量。



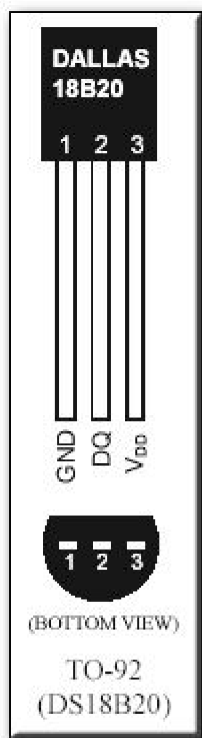


图 1-16　DS18B20 温度传感器

无线遥控接收板，接收模块有 7 根引脚，分别为 VCC、D3、D2、D1、D0、VT、GND，其中 VCC 为 5V 供电端，GND 为接地端，VT 为解码有效输出端，D3、D2、D1、D0 为 4 位数据锁存输出端，有信号时能输出 5V 左右的高电平，驱动电流约为 2mA，与发射器上的 4 个按键一一相对应，天线是一根长

度为 23cm 的软导线。无线发射器和接收模块如图 1-17 所示。



图 1-17　200m 遥控无线收发模块

如需本书配套的实验器材设备，可以与我们取得联系。网站上有大量的实验实例和视频演示录像供读者朋友学习参考，并且定期更新。

1.4　单片机学习的有效方法与途径

学习单片机是否会很困难呢？经常听到有人说，自己看了很多书，理论知识也灌了不少，但就是还不知道单片机是怎么回事，更谈不上去用它，感觉一团迷雾。其实，对于已经具有电子电路基础，尤其是数字电路基础知识的人来说，是不会有太大困难的。假如你对 PC 有一定的基础，学习单片机就会更容易些，毕竟有些原理还是相通的。但学好单片机最有效的方法与途径的关键还是在于能否将理论与实践相结合地去学习，应边看书，边动手，边实践，这样才能对理论知识得以深刻地理解与掌握。

目前，PC 的普及大大提高了单片机学习的系统环境，只要接上相应的开发设备，如仿真器、编程器就能进行编程学习及产品的开发，配上实验板进行实验，通过眼睛看到及耳朵听到的，更能给人一种直观的认识与体会，知道自己应该如何去编程以及编什么样的程序才能转化所需要的电信号去控制各类电子产品。

目前，单片机的种类非常多，如 MCS-51、AVR、PIC 等。MCS-51 系单片机是 Intel 公司的产品，它的生产量大，技术已相当成熟，学习资料丰富，价格也比较低廉，是最具代表性的一款单片机。因此，51 系列单片机对于广大初学者入门还是很有益的。本书中，我们讲

的内容是以 MCS-51 系列单片机为例，实验中所用到的烧写芯片为 Atmel 公司的 AT89S51 单片机，该单片机的性价比高。

本书突破传统教科书那种“教条式”的学习模式，根据作者的一些实践经验，采用边学理论，边实践验证的学习模式；按深入浅出、避繁就简、理论联系实际的原则，让你逐步对单片机基础知识及各方面的应用有所了解并掌握，一步一步地伴你跨入单片机技术的大门。

如需本书配套的实验器材设备，可以访问网站http：//www. hificat. com 了解详细信息，或与我们取得联系，联系电话：13185018567。网站上有大量的实验实例、视频演示录像供读者朋友学习参考，并且学习教程、资料定期不断更新。

第 2 章　单片机基础知识

本章将系统地介绍 51 单片机的内部结构、引脚定义和特性、存储器、寄存器等内容，但却并不想很深入地去探究它。对于起步阶段，过多地了解太细节的东西，反而阻碍了学习进度，尤其是在使用高级语言的前提下。在以后的学习和开发过程里，需要的时候，完全可以参考其他讲解更详细的书籍或者直接查看相应厂商的芯片资料。

2.1　MCS-51 单片机内部结构

2.1.1　MCS-51 单片机组成框图

目前有很多厂商生产 51 单片机，图 2-1 只是列出了几种比较常见的类型，从表面到内部资源不完全一样，但是它们的 MCU 结构确实是一致的，即都采用了 8051 核。

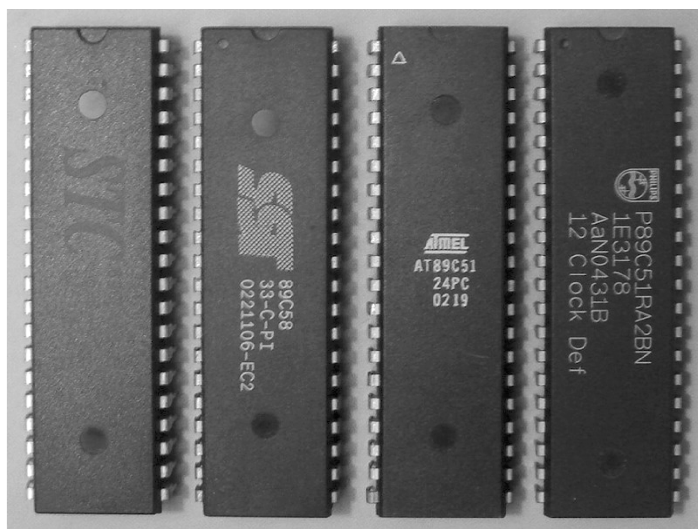


图 2-1　目前比较常用的几种 MCS-51 单片机外形

一个基本的 MCS-51 单片机通常包括如下一些部件：中央处理器、ROM、RAM、I/O 口、定时器、串口、中断控制器、振荡电路等。其简单的内部框图如图 2-2 所示。

并不是所有的 8051 单片机都具有图 2-2 所示的所有部件，当然也有比这个更多的，但是，图 2-2 所示的是个比较完整的、具有代表性的构成。其中 I/O 口、定时器、串口、中断控制器等是外部功能部件，就程序的执行来讲，单片机最离不开的部件应该是中央处理器、ROM、RAM、振荡电路这几个部分。

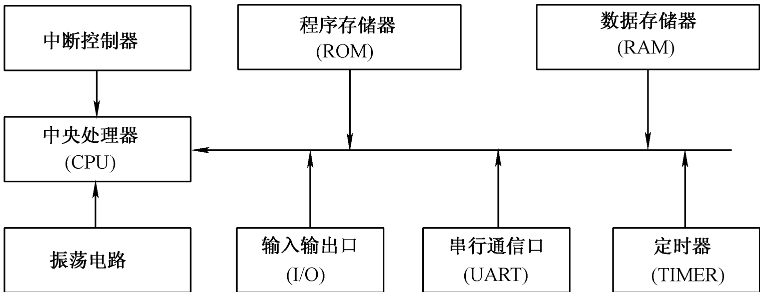


图 2-2　MCS-51 单片机基本组成框图

2.1.2　MCS-51 单片机工作机制

笔者认为，在了解每个部分的功能之前，如果对单片机本身的工作机制有比较多的了解，那么对理解每个部分的功能和存在的必要性会有比较清楚的认识。

首先明确一点，单片机是一个受程序控制的控制器，并且在你眼里看起来可能是同时发生的事情，事实上总是有先后的，因为单片机永远是按照特定的顺序来执行操作的，在指令层上只会在完成某个步骤之后，才开始另外一个步骤，无法同时处理两个操作。

在不考虑是否存在这样的工作的可能性，并且工人会严格按照顺序执行命令的前提下，假如有如图 2-3 所示的这样一个工作，那么我们或许可以像下面这样来描述单片机的工作过程：工人上班了，开始一天的工作；打开第一个抽屉，拿到的纸条上说“去看第三个抽屉”，所以工人就去开第三个抽屉了，便不用看第二个抽屉了；拿到第三个抽屉的纸条上说“工作台 1 号位置有没有东西？如果没有就去看第五个吧”；于是看了一下，有东西的，所以不能直接去看第五个了，而是按照顺序去拿第四个抽屉里的纸条；第四个纸条上写着：“把 1 号位置上的东西扔掉吧！”，所以 1 号位置上的东西就被扔掉了；接着是到第五个抽屉，收到命令说：“没事了，你等着吧”，于是就可以看报纸等下班了。

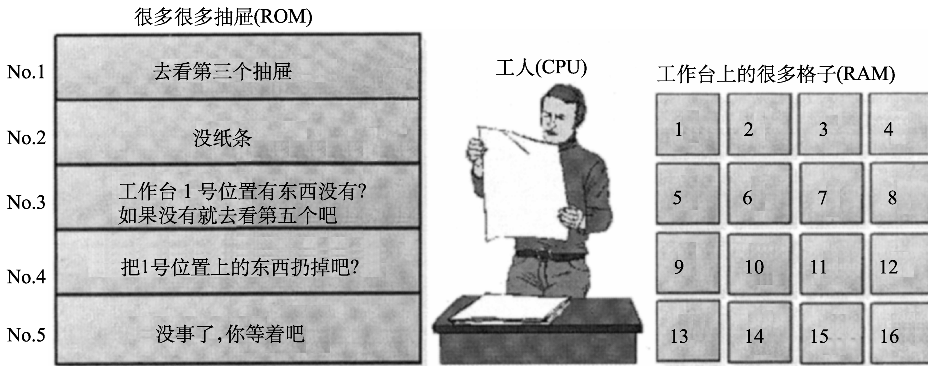


图 2-3　单片机工作机制模拟

虽然上述是个很简单、比较抽象的过程，但实际中单片机的工作机制确实很像刚才描述的过程，如果没有中断控制器的话，放在 ROM 里的程序就只会是一个不断循环的查询过程，由于程序执行速度很快，所以相对于人的反应时间来讲，很多操作感觉就像是实时的。再次强调一下，单片机一定是按照某种特定的顺序来执行指令的，除非程序本身改变了执行顺序

或者受到了很强的干扰。例如在单片机内部，当上电复位之后，取指令总是从 0000H 地址开始的，而地址的增量取决于指令的长度，如果是单字节长度的指令总是顺序执行程序，那么地址总是每次都自动加 1。

2.1.3 MCS-51 单片机内部功能部件

在第一小节提到：单片机最离不开的部件是中央处理器、ROM、RAM、振荡电路等部件，在第二小节中又把这几个部分分别看成是抽屉、工作台、工人，虽然比喻不是非常的确切，但是很能够说明问题，下面对每个部件的功能进行说明。

1. 只读存储器（ROM）

写出来的程序编译成最终的目标代码，通过烧写，被放在这里。代码是能够被识别的命令的序列（也可以有数据，一般是常量），用来指导 CPU 一步一步地去做事情。而写程序的你，就是领导了，就是那个往抽屉里放纸条的人，只不过纸条上写的东西，不是简单的一句“去看第三个抽屉”，而你现在在学的，就是怎么当领导。同时请注意“只读”的含义，对一个一般的程序执行过程来讲，程序是不可更改的，就是说，工人是不可以往抽屉里放纸条的。但在实际的系统中，有很多数据是变化的，因此单片机里还需要有一种存储器，就是 RAM，也就是程序执行的时候可以使用的工作台。

2. 随机存取存储器（RAM）

根据 MCS-51 单片机的结构特点，RAM 里放的肯定是数据，之所以叫随机存取存储器，是因为在工作过程中，数据可以随时读取和修改，正因为这样，一般而言在 C51 语言中定义的变量实际总会被定位在这里。对很多内部处理来说，RAM 的确很像工作台，用来暂时存放和处理一些数据。

3. 中央处理器（CPU）

虽然每个部分都不可缺少，但是在单片机内部，最重要的是中央处理器，它负责指令的读取、译码和执行等内部控制以及算术逻辑运算，当然它的结构也很复杂，由于是采用了高级语言来设计程序，我们就不必了解它，在写程序的时候你往往是感觉不到它的存在的，当然如果用汇编语言写的话就不同了。

4. 振荡电路

工人是需要吃饭的，但是单片机不是，推动单片机有条不紊地工作的动力在哪里呢？就是要说的振荡电路。振荡电路给出的时钟信号，使得由一大堆数字电路构成的单片机各个部件能够协同工作，并最终实现需要的功能。

有了前面说到的功能部件，程序已经可以执行了，但是如果仔细看看，其实这样的单片机是没有什么用的，首先数据从哪里来呀，捣鼓完之后的数据又有什么用呀，所以要构成实用的系统，还需要其他的部件。

5. 输入/输出（I/O）口

输入/输出口是单片机最普通也是最常用的部件，它可以用来获取外部的数字量，输出内部的数字量。例如通过指令可以获取当前 P0 口所有口线的状态，也可以通过指令控制口线输出高低电平，从而驱动连接在相应口线上的执行、指示部件产生动作，例如控制继电器、发光二极管等。

6. 定时/计数器

如果有个功能，需要它每隔100ms运行一次，并且对时间间隔要求比较高，这个时候如果仍然是用查询的方式，可能就达不到预期的效果。因为随着条件的变化，程序循环执行的周期有很大的不确定性，尤其是在程序量很大的情况下，这个时候用定时器配合中断方式来做，就比较合适了。定时器是个硬件计数器，简单地说就是有数脉冲个数的能力，当计数源是从外部输入的时候，它被称做计数器。

7. 中断控制器

读书的时候，老师给我们举例来说明中断的概念，他说，下课铃声就是个中断，因为睡着了的学生都会醒过来。严格地说，不仅仅是中断，还是中断唤醒，一个现在流行的低功耗MCU差不多都具有的功能。中断是指停止一件正在做的事情，然后处理另外那件突发事件，做完之后再回来接着做原来的事情。中断控制器就是用来控制单片机处理类似这样的问题的，它具有中断控制，中断优先级处理等功能。

2.2 引脚定义与特性

MCS-51系列单片机最常见的封装为40引脚双列直插式，其中许多引脚都具有第二功能，尤其是P3口。其引脚如图2-4所示，它的引脚按照功能可以分为4类：电源类引脚2个，时钟类引脚2个，并行I/O类引脚32个，控制类引脚4个。

1. 电源类引脚

V_{CC}（引脚40）：电源输入引脚，输入额定的工作电压；

GND（引脚20）：电源接地引脚。

2. 时钟类引脚

XTAL1（引脚19）：振荡电路输入端，连晶体振荡器；

XTAL2（引脚18）：振荡电路反相输出端，连晶体振荡器，同时也是外部时钟的输入端。

3. 并行I/O类引脚

P0口（引脚32~39）：普通I/O口，或在总线方式时作为地址（低8位）数据复用口，作为普通I/O口使用时无内部上拉电阻；

P1口（引脚1~8）：普通I/O口；

P2口（引脚21~28）：普通I/O口，或在总线方式时作为地址（高8位）；

P3口（引脚10~17）：普通I/O口，或作为其他第二功能口线，如串口、中断等。

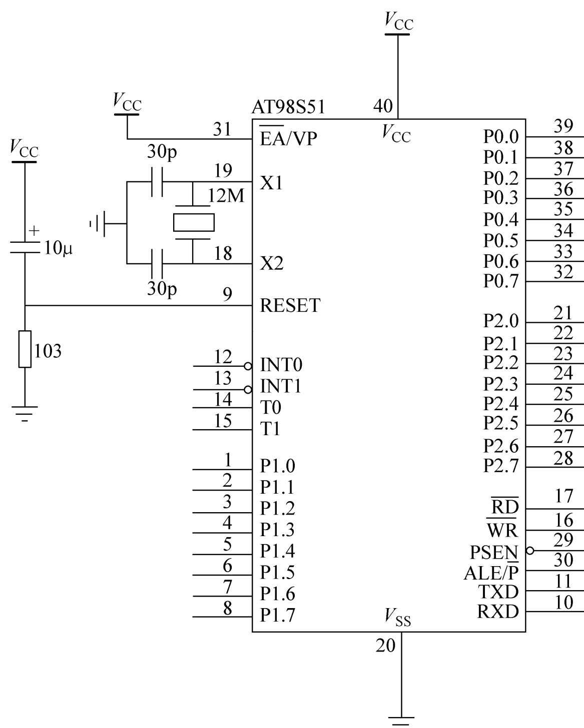


图2-4　最小系统原理

4. 控制类引脚

RST（引脚 9）：芯片复位引脚，持续时间超过两个时钟周期的高电平引起系统复位；

EA（引脚 31）：外部访问控制引脚，EA = 1 时程序从内部地址开始执行，否则从外部地址开始；

PSEN（引脚 29）：程序存储器访问使能；

ALE/P（引脚 30）：地址锁存允许输出信号，在总线方式时用来锁存低 8 位地址。

以上简要说明了各个引脚的作用，在实际添加外围电路的时候，引脚的驱动能力是个需要注意的问题，例如当驱动能力为 5～10mA 时，可以驱动一个发光二极管，但是不可以用来驱动继电器。如图 2-4 所示的电路，是个可以跑程序的最小系统，而在构建应用系统时，如果程序无法运行，说明问题基本上出现在这部分电路里，可以试着从下面几个方面检查：

1）电源是否已加上，用万用表检测引脚 40 和引脚 20 之间的电压，确保电源电路工作正常；

2）晶振是否已启动，用示波器检测引脚 18 和引脚 19 之间的波形，排除因虚焊等可能产生的问题；

3）确认 EA（引脚 31）连接正确，该引脚悬空可能导致工作不稳定；

4）检查 RST（引脚 9）是否为低电平，如果不是，则检查复位电路。

2.3 MCS-51 单片机存储器和寄存器

单片机要能够正常工作，在存储器方面，ROM 和 RAM 是不可以缺少的。同时，要控制很多部件，还需要更多的寄存器，因为很多部件都是和寄存器关联的。例如想要操作 P1 口，只需要对位于 90H 的 P1 寄存器操作就可以了，下面就主要来看看这两个方面。

2.3.1 MCS-51 单片机的存储器结构

目前，很多单片机一般都会在内部集成一定数量的程序存储器和数据存储器，而且根据程序量的大小，都可以在一个系列里找到相应的型号，所以通常都不再需要像以前的 8031 那样扩展外部程序存储器，下面以 AT89C51 为例来看看存储器的结构，如图 2-5 所示。

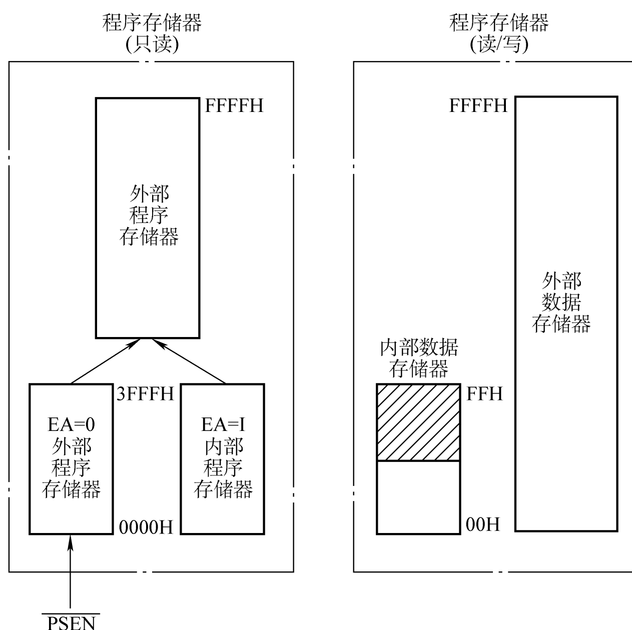


图 2-5　AT89C51 存储器的结构

就程序存储器而言，AT89C51 已经集成了 4KB，一般的应用已经足够。但是，如果在外部扩展到 64KB 的话，那么其分布就如图 2-5 左边所表示的一样。如果 EA = 0，那么程序的执行全部在外部空间，反之如果 EA = 1，那么当程序小于 4KB 的时候，程序执行在内部空间，而大于 4KB 时，超出部分就

会到外部空间。总之，程序存储器有部分重叠。

数据存储器也分成内外两种，由于采用不同的方法访问，所以不存在重叠的现象。一般如果不是设计得比较复杂的数据采集系统，扩展外部 RAM 的可能性也比较小，所以比较重要的是内部的 RAM 空间，说它重要其实还有另外一个原因：特殊功能寄存器都分布在这里。

内部 RAM 分成两大部分，即普通 RAM 和特殊功能寄存器，其地址范围分别为 00H ~ 7FH 和 80H ~ F0H。普通 RAM 的具体分类如图 2-6 所示，特殊功能寄存器的示意图如图 2-7 所示。

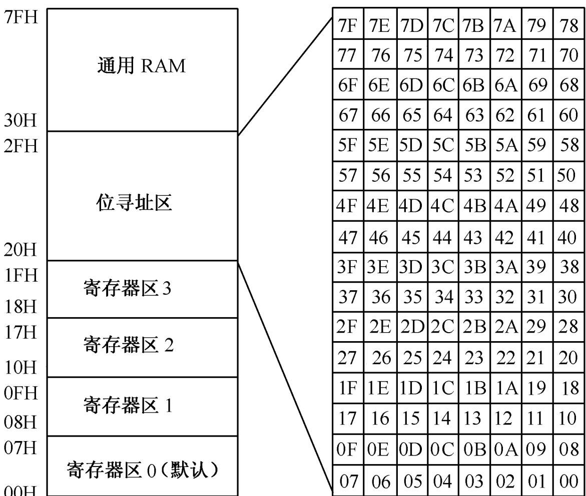


图 2-6　普通 RAM 区结构

虽然出于功能区分，这些 RAM 被分成很多类别，但它们在本质上是一样的，不过是个“内容可以修改的地址”而已，所以很多时候，对上述的通用寄存器区和特殊功能寄存器，除了可以用特殊的名字，例如 P0 进行访问之外，也完全可以用其地址 80H 进行直接访问。这里还有一个很特殊的 RAM 区，即可位寻址 RAM 区，其特点是一个字节中的每个 bit 都是可以单独访问的，这些数据对应了 C51 语言里的一种扩展的数据类型 bdata，在以后的程序设计中，肯定会用到。

SCON	98H	···	
···		B	F0H
P1	90H	···	
···	00H	ACC	E0H
TH1	8DH	···	
TH0	8CH	PSW	D0H
TL1	8BH	···	
TL0	8AH	IP	B8H
TMOD	89H	···	
TCON	88H	P3	B0H
PCON	87H	···	
···		IE	A8H
DPH	83H	···	
DPL	82H	P2	A0H
SP	81H	···	
P0	80H	SBUF	99H

图 2-7　特殊功能寄存器结构

2.3.2　MCS-51 单片机的寄存器

MCS-51 单片机的寄存器分成两类，一种是通用寄存器，前面提到的通用寄存器区便是这个类别的，但是在采用高级语言编程时，这些寄存器其实是看不到的，通常它们会被用来

作为函数内部的局部变量和函数调用时传递参数之用，由编译器统一接管。但是作为特殊功能寄存器的另外一种寄存器，是需要程序中处理的，因为他们往往对应了硬件操作，需要开发人员按功能需求进行控制。所以粗略地了解一下这些寄存器的功能，需要的时候可以翻书来查看。

1. P0、P1、P2、P3

这4个特殊功能寄存器对应了4组I/O口，通过对这些地址的读和写，可以改变和获得I/O的状态。例如下面的程序，就完成了对P1口的读取和P3口的输出操作。

```
unsigned char temp;
temp  =  P1;                  // 读取 P1 口的数据到 temp
P3  =  temp;                  // 把 temp 数据从 P3 口输出
```

如果需要对单独的某个I/O口线进行操作，可以参考下面的程序。

```
sbit P1_1 = P1^1;
P1_1  =  1;                   // 把 P1.1 设置成高电平
P1_1  =  0;                   // 把 P1.1 清零为低电平
```

2. PCON

这个寄存器叫做电源控制寄存器，主要用来作功率控制之用。所谓功率控制，主要是指在那些要求低功耗的场合，例如某个设备如果5min没有操作便自动进入待机模式，这是出于节能的目的，使单片机进入掉电或空闲模式。这个寄存器中实际与这个功能相关的bit只有两个：PD和IDL，即掉电控制位和空闲模式位。最高位SMOD是串口波特率加倍控制位，置1时，串口波特率加倍。GF1和GF0是通用的标志位。

SMOD	——	——	——	GF1	GF0	PD	IDL

3. TMOD、TCON、TH1、TH0、TL1、TL0

这组寄存器全部和定时/计数器相关，其中TMOD为模式控制寄存器，用来设定工作模式；TCON为控制寄存器，用来对定时/计数器进行开启/关闭和终端势能等的控制。THn和TLn分别是相应的计数寄存器。

TMOD	GATE	C/T	M1	M0	GATE	C/T	M1	M0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE1	IT0

4. SCON、SBUF

这两个寄存器和串口相关，SCON是串行通信口的控制寄存器，完成波特率以及数据格式的设定和中断使能控制等；SBUF则是数据收发的缓冲区。

5. IE、IP

这两个寄存器控制了中断功能的使能以及优先级别的设定。其中IE.7为EA，是所有中断的总控制开关，其他5个bit则是对应中断的控制开关，对应的是IP中这5个中断的优先级设置位。

IE	EA	——	——	ES	ET1	EX1	ET0	EX0
IP	——	——	——	PS	PT1	PX1	PT0	PX0

6. SP、PSW、ACC、B、DPH、DPL

这些寄存器对单片机的工作而言是很重要的，但是因为用了 C51 来设计程序，对程序设计来说，它们就不再是非了解不可了。

2.4　定时/计数器

2.4.1　定时/计数器概述

定时/计数器是单片机系统的一个重要部件，可以用来实现定时控制、频率测量、脉宽测量、信号发生等。此外，定时/计数器还可以用来作为串行通信中的波特率发生器。

在 MCS-51 单片机中采用加法计数器，先设置计数器的初值，然后对计数脉冲每次加一，加到计数器溢出为止。这就是所谓的加法计数器。

定时/计数器有定时和计数两大功能，但归根到底是一个计数器。对外部事件脉冲计数是计数器，对片内机器周期脉冲计数是定时器。在日常生活中也常用到定时的概念，如将时钟定时到 1min，那么秒钟计时到 60 后就到了 1min 的定时。单片机内部工作原理也类似。

2.4.2　定时/计数器结构

MCS-51 单片机内部设置两个 16 位可编程的定时/计数器 T0 和 T1，其结构如图 2-8 所示，它们具有计数器方式和定时器方式以及 4 种工作模式。其状态字均在相应的特殊功能寄存器中，通过对控制寄存器的编程，可以方便地选择适当的工作模式。对每个定时/计数器，在特殊功能寄存器 TMOD 中都有一个控制位，它选择 T0 及 T1 为定时器或计数器。定时器 T0 由特性功能寄存器 TL0（低 8 位）和 TH0（高 8 位）构成，定时器 T1 由特性功能寄存器 TL1（低 8 位）和 TH1（高 8 位）构成。特殊功能寄存器 TMOD 控制定时寄存器的工作方式，TCON 则用于控制定时器 T0 和 T1 的启动和停止计数，同时管理定时器 T0 和 T1 的溢出标志等。程序开始时需对 TL0、TH0、TL1 和 TH1 进行初始化编程，以定义它们的工作方式

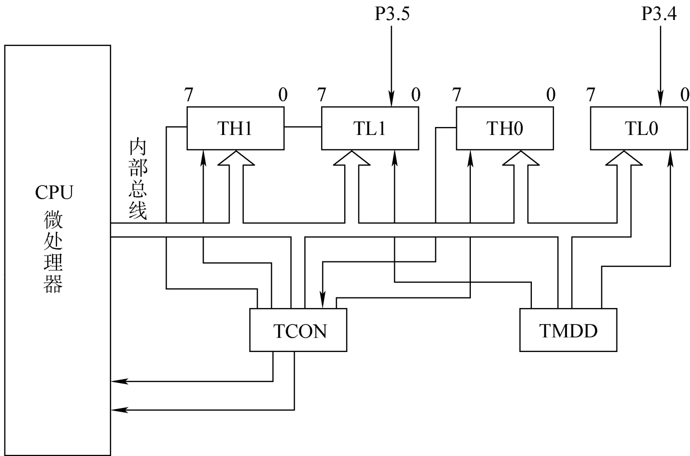


图 2-8　MCS-51 内部定时/计数器结构

和控制 T0 和 T1 的计数。系统复位时，寄存器的所有位都被清零。

2.4.3 定时/计数器控制寄存器

在单片机中有两个特殊功能寄存器与定时/计数器有关，其编程操作通过两个特殊功能寄存器 TMOD 和 TCON 的状态设置来实现。特殊功能寄存器 TMOD 控制定时器的工作方式，TCON 控制其运行，TCON 还包含 T0 和 T1 的溢出标志。

1. 定时/计数器控制寄存器 TCON

TCON 的字节地址为 88H，每一位有位地址，均可位操作。TCON 寄存器已在中断系统中介绍过，它的低 4 位只与中断有关，此处不再重复，高 4 位与定时/计数器有关。TCON 的结构和各位名称、功能见表 2-1。

表 2-1 TCON 结构

TCON 结构							
D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：定时/计数器 T1 溢出标志。当 T1 被允许计数后，T1 从初值开始加 1 计数，至最高位产生溢出时，TF 置“1”，既表示计数器溢出又表示请求中断。CPU 响应中断后由硬件自动对 TF1 清零。

TR1：定时/计数器 T1 运行控制位。靠软件置位或清除，当 TR1 = 1 时启动 T1 运行，TR1 = 0 时则 T1 停止运行。

TF0：定时/计数器 T0 溢出标志。其意义与 TF1 相似。

TR0：定时/计数器 T0 运行控制位。其意义与 TR1 相似。

2. 定时/计数器工作方式寄存器 TMOD㊀

TMOD 用于设定定时/计数器的工作方式和工作模式。低 4 位用于 T0，高 4 位用于 T1，TMOD 的结构和各位名称、功能见表 2-2。

表 2-2 TMOD 寄存器结构

TMOD 寄存器结构							
D7	D6	D5	D4	D3	D2	D1	D0
GATA	$C/\overline{T}$	M1	M0	GATA	$C/\overline{T}$	M1	M0
←T1 方式字段→				←T0 方式字段→			

M1M0：工作方式选择位，两位二进制位可表示 4 种状态，具体功能见表 2-3。

表 2-3 工作方式选择

M1M0	工作方式	功　　能
00	方式 0	13 位计数器
01	方式 1	16 位计数器
10	方式 2	两个 8 位计数器，初值自动装入
11	方式 3	两个 8 位计数器，仅适用于 T0

㊀ TMOD 只能按字节寻址。

$C/\overline{T}$：计数/定时方式选择位。

$C/\overline{T}=1$ 时为计数方式，对外部事件脉冲计数，负跳变脉冲有效。

$C/\overline{T}=0$ 时为定时工作方式，对片内机器周期脉冲计数，用作定时器。

GATE：门控位。

GATE＝0 时定时/计数器的运行只受 TCON 中运行控制位 TR 的控制。

GATE＝1 时定时/计数器的运行同时受 TR 和外中断输入信号的双重控制。

2.4.4 定时/计数器的工作方式

在 TMOD 控制寄存器中可知对 M1M0 的不同设置可选择 4 种不同工作方式，下面以 T0 为例分别对这 4 种工作方式作一介绍。

1. 工作方式 0

定时/计数器 0 的工作方式 0 的电路逻辑结构如图 2-9 所示（定时/计数器 1 与其完全一致）。工作方式 0 是 13 位计数结构的工作方式，其计数器由 TH 的全部 8 位和 TL 的低 5 位构成，TL 的高 3 位没有使用。当 $C/\overline{T}=0$ 时，多路开关接通振荡脉冲的 12 分频输出，13 位计数器依次进行计数。这就是定时工作方式。当 $C/\overline{T}=1$ 时，多路开关接通计数引脚（T0），外部计数脉冲由引脚 T0 输入。当计数脉冲发生负跳变时，计数器加 1，这就是我们常说的计数工作方式。

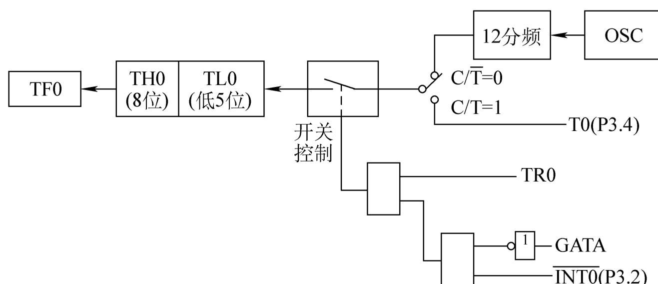


图 2-9 工作方式 0

不管是哪种工作方式，当 TL 的低 5 位溢出时，都会向 TH 进位，而全部 13 位计数器溢出时，则会向计数器溢出标志位 TF0 进位。

如上所述，TF0 是定时/计数器的溢出状态标志，溢出时由硬件置位，TF0 溢出中断被 CPU 响应时，转入中断时硬件清零，TF0 也可由程序查询和清零。

2. 工作方式 1

当 M1M0＝01 时，定时/计数器处于工作方式 1，此时，定时/计数器的等效电路如图 2-10 所示。

方式 0 和方式 1 的区别仅在于计数器的位数不同，方式 0 为 13 位，而方式 1 则为 16 位，由 TH0 作为高 8 位，TL0 为低 8 位，有关控制状态字（GATA、$C/\overline{T}$、TF0、TR0）和方式 0 相同。

3. 工作方式 2

当 M1M0＝10 时，定时/计数器处于工作方式 2。此时定时器的等效电路如图 2-11 所示。

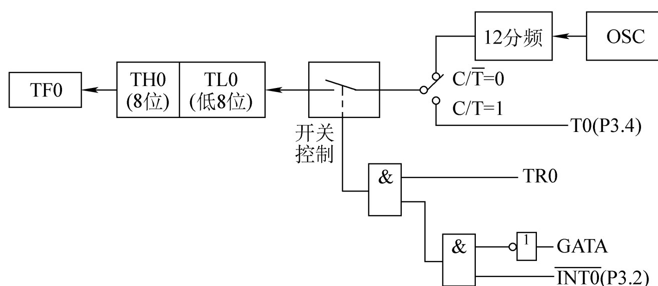


图 2-10　工作方式 1

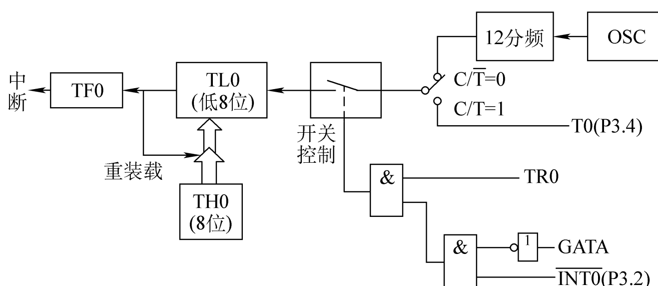


图 2-11　工作方式 2

定时/计数器 1 与之完全一致。

工作方式 0 和工作方式 1 的最大特点就是计数溢出后，计数器为全 0，因而循环定时或循环计数应用时就存在反复设置初值的问题，这给程序设计带来许多不便，同时也会影响计时精度。工作方式 2 就针对这个问题而设置，它具有自动重装载功能，即自动加载计数初值，所以也有的文献称之为自动重加载工作方式。在这种工作方式中，16 位计数器分为两部分，即以 TL0 为计数器，以 TH0 作为预置寄存器，初始化时把计数初值分别加载至 TL0 和 TH0 中，当计数溢出时，不再像方式 0 和方式 1 那样需要“人工重复加载"，由软件重新赋值，而是由预置寄存器 TH 以硬件方式自动给计数器 TL0 重新加载。

程序初始化时，给 TL0 和 TH0 同时赋以初值，当 TL0 计数溢出时，置位 TF0 的同时把预置寄存器 TH0 中的初值加载给 TL0，TL0 重新计数。如此反复，这样省去了程序不断需给计数器赋值的麻烦，而且计数准确度也提高了。但这种方式也有其不利的一面，即计数结构只有 8 位，计数值有限，最大只能到 255。所以这种工作方式很适合于那些重复计数的应用场合。例如我们可以通过这样的计数方式产生中断，从而产生一个固定频率的脉冲。也可以当作串行数据通信的波特率发送器使用。

4. 工作方式 3

当 M1M0 = 11 时，定时/计数器处于工作方式 3。此时定时/计数器的等效电路如图 2-12 所示。仍以定时器 0 为例，值得注意的是，在工作方式 3 模式下，定时/计数器 1 的工作方式与之不同。

在工作方式 3 模式下，定时/计数器 0 被拆成两个独立的 8 位计数器 TL0 和 TH0。其中 TL0 既可以作计数器使用，也可以作为定时器使用，定时/计数器 0 的各控制位和引脚信号全归它使用。其功能和操作与方式 0 或方式 1 完全相同。但 TH0 就只能作为简单的定时器使用，而且由于定时/计数器 0 的控制位已被 TL0 占用，因此只能借用定时/计数器 1 的控制

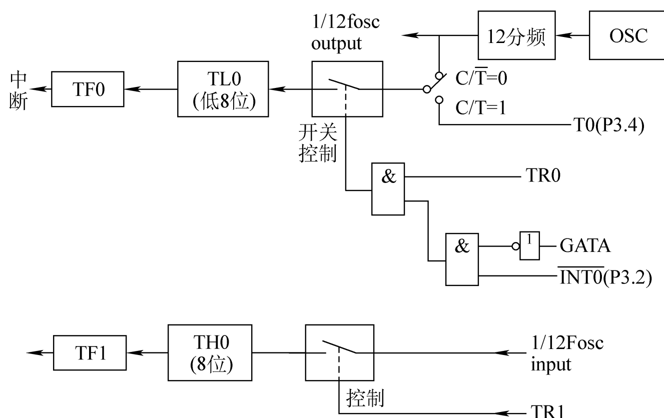


图 2-12　工作方式 3

位 TR1 和 TF1，也就是以计数溢出去置位 TF1，TR1 则负责控制 TH0 定时的启动和停止。由于 TL0 既能作定时器也能作计数器使用，TH0 只能作定时器使用而不能作计数器使用，因此在方式 3 模式下，定时/计数器 0 可以构成两个定时器或者一个定时器和一个计数器。

2.4.5　定时/计数器的应用

1. 定时/计数器初始化

定时/计数器初始化过程如下：

1）根据要求给方式寄存器 TMOD 送一个方式控制字，以设定定时器响应的工作方式；

2）根据需要给 $C/\overline{T}$ 选送初值，以确定需要的定时时间或计数的初值；

3）根据需要给中断允许寄存器 IE 送中断控制字，以开放相应的中断和设定中断优先级；

4）给 TCON 送命令字，以启动或禁止 $C/\overline{T}$ 运行。

2. 定时/计数器初值的计算

MCS-51 单片机定时/计数器初值计算公式为

$$T(\text{初值}) = 2^N - \text{定时时间} / \text{机器周期时间}$$

式中，N 与工作方式有关。方式 0 时，$N = 13$；方式 1 时，$N = 16$；方式 2 和 3 时，$N = 8$。机器周期时间 $= 12/f_{osc}$。

【例 2-1】　已知晶振为 12MHz 时，求定时 0.2ms 时，T0 工作方式 0、方式 1、方式 2、方式 3 时的定时初值。

（1）工作方式 0

$2^{13} - 200/1 = 8192 - 200 = 7992 = 1F38H$。

1F38 化成二进制：1F38 = 0001 1111 0011 1000 B

则低 5 位送 TL0 为 18H，高 8 位送 TH0 为 F9H。

（2）工作方式 1

$2^{16} - 200/1 = 65536 - 200 = 65336 = FF38H$。即 TH0 = FFH，TL0 = 38H。

（3）工作方式 2

$2^{8} - 200/1 = 256 - 200 = 56 = 38H$。即 TH0 = 38H，TL0 = 38H。

（4）工作方式 3

方式 3 与方式 2 初值一样为 TH0 = TL0 = 38H。

2.4.6 定时器的应用

设单片机系统时钟频率为 12MHz，要求在 P0.0 的 LED 定时 50ms 循环亮灭。硬件原理如图 2-13 所示。

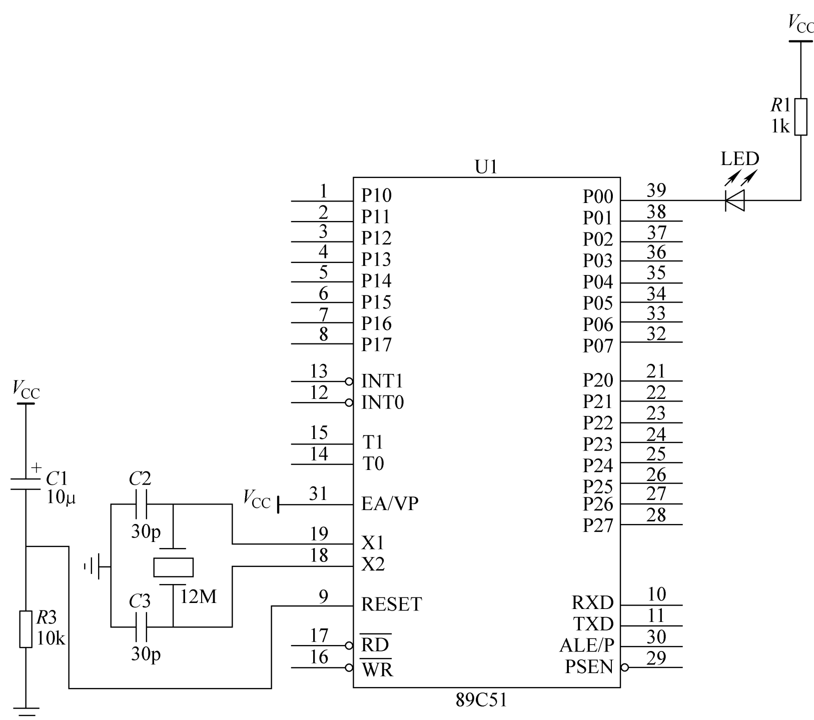


图 2-13 硬件原理

程序设计：

定时器初值计算：T0 初值 = 2^{16} − 50000μs/1μs = 65536 − 50000 = 15536 = 3CB0H，则 TH0 = 3CH，TL0 = B0H。在 C 语言中对定时器初始化还可以用更简单的方法，即直接用减初值的方法，如定时 50ms 在语句中可以写成

```
TH0 = -50000/256;                    //定时器 T0 的高四位赋值
TL0 = -50000%256;                    //定时器 T0 的低四位赋值
```

完整程序如下：

```
#include <reg51.h>
#include <absacc.h>

sbit LED = P0^0;

void main (void)
{
P0 = 0xff;                           // 初始化端口
```

```
EA =1;                                    //允许所有中断
ET0 =1;                                   //允许 T0 中断
TMOD =0x01;                               //T0 方式 1 计时 0.05s
TH0 = -50000/256;                         //定时器 T0 的高四位赋值
TL0 = -50000%256;                         //定时器 T0 的低四位赋值
TR0 =1;                                   //开中断，启动定时器
for ( ;;);
}
/* 定时计数器 0 的中断服务子程序 */
void intserv1 (void) interrupt 1 using 1
{
TH0 = -50000/256;                         //定时器 T0 的高四位赋值
TL0 = -50000%256;                         //定时器 T0 的低四位赋值
LED =! LED;
}
```

2.5 MCS-51 单片机中断系统

什么是中断？对初学者来说，中断这个概念比较抽象，其实单片机的处理系统与人的一般思维有着许多相似之处，在日常生活和工作中有很多类似的情况。假如你正在写文章，这时候手机响了，你在文稿上做个记号，然后与对方通电话，通话完毕，继续写你的文章。这就是生活中的“中断”的现象，就是正常的工作过程被突然发生的事件打断了。

2.5.1 单片机中断

对于单片机中的中央处理器处理事情也和我们写文章、接电话一样，单片机中 CPU 只有一个，但在同一时间内可能会面临着处理很多任务的情况，如运行主程序、数据的输入和输出、定时/计数时间已到、可能还有一些外部的更重要的中断请求要先处理。此时也得像人的思维一样停下某一样（或几样）工作先去完成一些紧急任务的中断方法。

仔细研究一下生活中的中断，对于我们学习单片机的中断也很有好处。首先，什么导致引起中断，生活中很多事件可以引起中断：有人按门铃了，电话铃响了，你的闹钟响了……等等诸如此类的事件，我们把可以引起中断的事件称之为中断源，单片机中也有一些可以引起中断的事件。

所以，中断是指计算机在执行某一程序的过程中，由于计算机系统内、外的某种原因，而必须中止原程序的执行，转去执行相应的处理程序，待处理结束之后，再回来继续执行被中止的原程序的过程。这样类似的处理方法上升到计算机理论，就是一个资源面对多项任务的处理方式，由于资源有限，面对多项任务同时要处理时，就会出现资源竞争的现象。中断技术就是为了解决资源竞争的一个可行的方法，采用中断技术可使多项任务共享一个资源，所以有些文献也称中断技术是一种资源共享技术。

2.5.2 中断的必要性

为什么要设置中断？设中断有什么优点和功能？采用了中断技术后的计算机，可以解决CPU与外设之间速度匹配的问题，使计算机可以及时处理系统中许多随机的参数和信息，同时，它也提高了计算机处理故障与应变的能力。

2.5.3 中断源

中断源是指能发出中断请求，引起中断的装置或事件。中断源主要有以下几种：

外设：如A/D转换器、打印机、键盘等。

故障源：如电源掉电、运算溢出、存储器损坏等。

定时器定时到预定的时间所发出的中断请求。

2.5.4 中断优先级

单片机系统的中断源一般不止一个，若在同一时刻有两个中断源同时向CPU请求中断，那怎么办？51单片机将5个中断源划分为两个中断优先级：高优先级和低优先级。中断优先级越高，则响应优先权就越高。当CPU正在执行中断服务程序时，又有中断优先级更高的中断申请产生，这时CPU就会暂停当前的中断服务转而处理高级中断申请，待高级中断处理程序完毕再返回原中断程序断点处继续执行，这一过程称为中断嵌套。

在默认情况下单片机系统本身也将所有中断源按优先权先后排列，若同一时间同时请求中断的两个中断源不属于同一优先级，则CPU先响应优先级高的中断源，等优先级高的中断响应完了之后再响应另一中断。

2.5.5 中断响应过程

CPU响应中断请求后，就立即转入执行中断服务程序。不同的中断源、不同的中断要求可能有不同的中断处理方法，但它们的处理流程一般都如下所述。

1. 保护现场

中断是在执行其他任务的过程中转去执行临时的任务，为了在执行完中断服务程序后，回头执行原来的程序时，知道原来的程序在何处打断的，各有关寄存器的内容如何，就必须在转入执行中断服务程序前，将这些内容和状态进行备份——即保护现场。

如果在执行中断服务时不是按上述方法进行现场保护和恢复现场，就会使程序运行紊乱，程序跑飞，自然使单片机不能正常工作。

在汇编语言中一般用如下语句保护现场：

```
PUSH    ACC
PUSH    PSW
PUSH    DPH
PUSH    DPL
```

而在C语言中程序会自动保护现场。在使用中不用对现场进行人为保护和恢复。

2. 中断服务程序

既然有中断产生，就必然有其具体的需执行的任务，中断服务程序就是执行中断处理的

具体内容，一般以子程序的形式出现，所有的中断都要转去执行中断服务程序，进行中断服务。

3. 中断返回

执行完中断服务程序后，必然要返回，中断返回就是程序从中断服务程序转回到原工作程序上来。同时，中断服务程序完成后，就需把保存的现场内容从堆栈中弹出，恢复寄存器和存储单元的原有内容，这就是现场恢复。在 MCS-51 单片机中，中断返回是通过一条专门的指令实现的，自然这条指令是中断服务程序的最后一条指令。在汇编语言中典型的中断返回语句如下（恢复现场的语句与保护现场的语句遵循后进先出规则）：

```
POP   DPL          恢复现场
POP   DPH
POP   PSW
POP   ACC
RETI               返回
```

2.6 中断系统

2.6.1 中断系统结构

8051 有 5 个中断源，它们是两个外中断 $\overline{INT0}$（P3.2）和 $\overline{INT1}$（P3.3）、两个片内定时/计数器溢出中断 TF0 和 TF1，一个是片内串行口中断 TI 或 RI，这几个中断源由 TCON 和 SCON 两个特殊功能寄存器进行控制。其结构如图 2-14 所示。

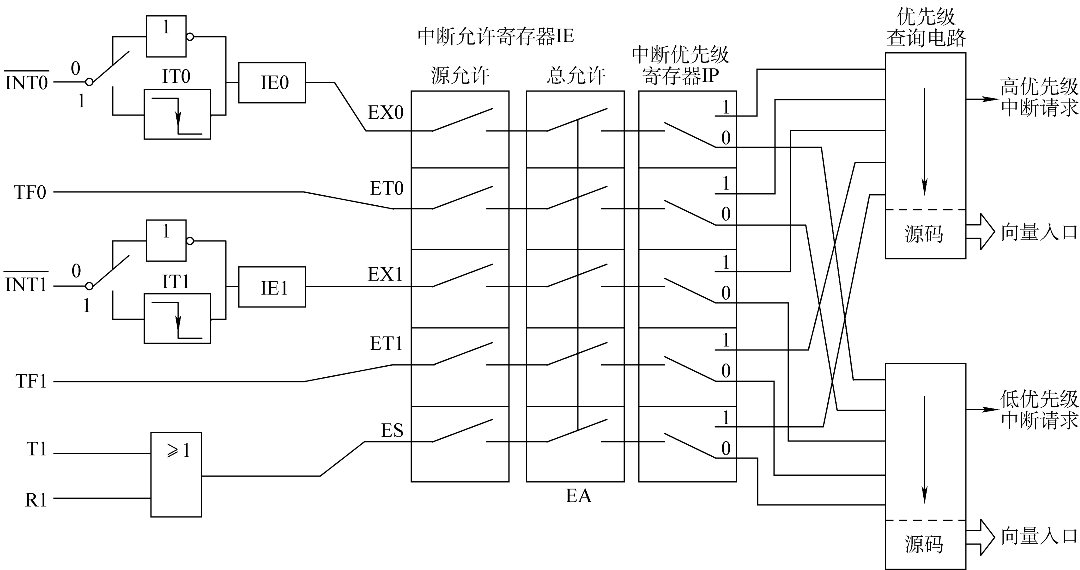


图 2-14 中断系统结构

计算机的中断系统能够加强 CPU 对多任务事件的处理能力。从而使它的应用范围进一步扩大。在 MCS-48 结构的基础上，MCS-51 在增强了 I/O 的种类、功能和数量的同时，也

增强了中断能力。MCS-51 提供了 5 个中断源，两个中断优先级控制，可实现两个中断服务嵌套。当 CPU 支持中断屏蔽指令后，可将一部分或所有的中断关断，只有打开相应的中断控制位后，方可接收相应的中断请求。程序设置中断的允许或屏蔽，也可设置中断的优先级。

2.6.2 MCS-51 中断源

MCS-51 单片机共有 5 个中断源，其中 2 个外部中断源，3 个内部中断源：

INT0：外部中断 0，中断请求信号由 P3.2 输入。

INT1：外部中断 1，中断请求信号由 P3.3 输入。

T0：定时/计数器 0 溢出中断。

T1：定时/计数器 1 溢出中断。

串行中断：包括串行接收中断 RI 和串行发送中断 TI。

2.6.3 中断控制

MCS-51 单片机对中断的控制主要有 4 个特殊功能寄存器：

定时/外部中断控制寄存器 TCON；串行中断控制寄存器 SCON；中断允许控制寄存器 IE；中断优先级控制寄存器 IP。

其中 TCON 和 SCON 只有一部分用于中断控制，通过以上 4 个中断控制寄存器各位进行配置，可实现各种中断控制功能。

1. 定时/外部中断控制寄存器 TCON（见表 2-4）

表 2-4 定时/外部中断控制寄存器 TCON

TCON	D7	D6	D5	D4	D3	D2	D1	D0
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H

其中断有关控制位功能如下：

IT0：外部中断 0 类型控制位，通过软件设置或清除，用于控制外中断的触发信号类型。当 IT0 =1 时是边沿触发，当 IT0 =0 时是电平触发。

IE0：外部边沿触发中断 0 请求标志，当引脚 P.2 信号有效时，触发 IE0 置“1”，当 CPU 响应该中断后，由片内硬件自动清零（只适用于边沿触发方式）。

IT1：外部中断 1 类型控制位，其意义和 IT0 一样。

IE1：外部边沿触发中断 0 请求标志，其意义和 IE0 一样。

2. 串行中断控制寄存器 SCON（见表 2-5）

表 2-5 串行中断控制寄存器 SCON

SCON	D7	D6	D5	D4	D3	D2	D1	D0
	—	—	—	—	—	—	TI	RI
位地址							99H	98H

各位功能如下：

RI：串行口接收中断标志。若串行口接收器允许接收，并以方式 0 工作，每当接收到 8

位数据时，RI 被置 1；若以方式 1、2、3 工作，当接收到半个停止位时，RI 被置 1；当串行口以方式 2 或 3 工作，且当 SM2 = 1 时，仅当接收到第 9 位数据 RB8 为 1 后，同时还要在接收到半个停止位时，RI 被置 1。RI = 1 表示串行口接收器正向 CPU 申请中断。同样 RI 标志必须由用户软件清零。

TI：串行口发送中断标志。在串行口以方式 0 发送时，每当发送完 8 位数据后由硬件置位。如果以方式 1、2、3 发送时，在发送停止位的开始时 TI 被置 1。TI = 1 表示串行发送器正向 CPU 发出中断请求，向串行口的数据缓冲器 SBUF 写入一个数据后就立即启动发送器继续发送。但是 CPU 响应中断请求后，转向执行中断服务程序时，并不清零 TI，TI 必须由用户的中断服务程序清零。

3. 中断允许控制寄存器 IE（见表 2-6）

表 2-6　中断允许控制寄存器 IE

IE	D7	D6	D5	D4	D3	D2	D1	D0
	EA	—	—	ES	ET1	EX1	ET0	EX0
位地址	AFH			ACH	ABH	AAH	A9H	A8H

其各位功能如下：

EX0：外中断 0 中断控制位。EX1 = 1，允许外中断 0 中断；EX1 = 0，禁止外中断 0 中断。

ET0：定时/计数器 T0 中断控制位。ET1 = 1，允许 T0 中断；ET1 = 0，禁止 T0 中断。

EX1：外中断 1 中断控制位。EX1 = 1，允许外中断 1 中断；EX1 = 0，禁止外中断 1 中断。

ET1：定时/计数器 T1 中断控制位。ET1 = 1，允许 T1 中断；ET1 = 0，禁止 T1 中断。

ES：串行口中断控制位。ES = 1，允许串行口中断；ES = 0，禁止串行口中断。

EA：中断总控制位。EA = 1，CPU 开放中断；EA = 0，CPU 禁止所有中断。

4. 中断优先级控制寄存器 IP（见表 2-7）

表 2-7　中断优先级控制寄存器 IP

IP	D7	D6	D5	D4	D3	D2	D1	D0
	—	—	—	PS	PT1	PX1	PT0	PX0
位地址				BCH	BBH	BAH	B9H	B8H

其各位功能如下：

PX0：外中断 0 优先级控制位。PX0 = 1，声明外中断 0 为高优先级中断；PX0 = 0，定义外中断 0 为低优先级中断。

PT0：定时器 0 优先级控制位。PT0 = 1，声明定时器 0 为高优先级中断；PT0 = 0，定义定时器 0 为低优先级中断。

PX1：外中断 1 优先级控制位。PX1 = 1，声明外中断 1 为高优先级中断；PX1 = 0，定义外中断 1 为低优先级中断。

PT1：定时器 1 优先级控制位。PT1 = 1，声明定时器 1 为高优先级中断；PT1 = 0，定义定时器 1 为低优先级中断。

PS：串行口中断优先级控制位。PS = 1，串行口中断声明为高优先级中断；PS = 0，串

行口定义为低优先级中断。

2.6.4 中断响应等待时间

正常中断响应时间至少为3～8个机器周期，如果有同级或高级中断服务，将延长中断响应时间。

2.6.5 中断撤消

中断源发出中断请求后，相应中断请求标志置“1”。CPU 响应中断后，必须清除中断请求标志，否则中断响应返回后将再次进入该中断，从而进入死循环。详细说明如下：

（1）对定时/计数器 T0、T1 中断，CPU 响应中断时就用硬件自动清除了相应的中断请求标志 TF0 或 TF1，用户可以不必手动清除。

（2）对外部中断 INT0、INT1，若采用边沿触发方式，CPU 响应中断时也用硬件自动清除了相应中断请求标志 IE0 或 IE1。

（3）对外部中断 INT0、INT1，若采用电平触发方式，CPU 响应中断时，虽也是用硬件自动清除了相应中断请求标志 IE0 或 IE1，但相应引脚的低电平信号若继续保持下去，中断请求标志 IE0 或 IE1 就无法清零，也会发生再次进入中断的情况。

（4）对串行中断，CPU 响应中断后并不自动清除相应的中断标志位 TI 或 RI，用户应在串行中断服务程序中用软件清除 TI 或 RI。

2.6.6 中断系统应用举例

C51 编译器支持在 C 语言源程序中直接编写 51 单片机的中断服务程序，比汇编编写中断程序方便许多。为了能在 C 语言中直接编写中断服务函数，C51 编译器对函数的定义有所扩展，增加了一个扩展关键字 interrupt。关键字 interrupt 是函数定义时的一个选项，加上这个选项即可将函数定义成中断服务函数。

定义中断服务函数的一般形式为

函数类型 函数名（形式参数）[interrupt n] [using n]

interrupt 后面的 n 是中断号，n 的取值范围为 0～31。编译器从 $8n+3$ 处产生中断向量，具体的中断号 n 和中断向量取决于不同的 51 系列单片机芯片。using n 中的 n 表示使用的寄存器组号。在 C 语言中可用如下方法定义中断服务函数：

void intersvr0（void）interrupt 0 using 1 //定义外部中断 0，使用第一组寄存器。

51 单片机常用的中断源和中断向量见表 2-8。

表 2-8 常用的中断源和中断向量

中断编号	中断源	入口地址
0	外部中断 0	0003H
1	定时/计数器 0 溢出中断	000BH
2	外部中断 1	0013H
3	定时/计数器 1 溢出中断	001BH
4	串行口中断	0023H

外部中断的应用范例：

硬件原理如图 2-15 所示。利用外部中断（S1）来控制 P00 的 LED，当有外部中断时 LED 的状态取反。

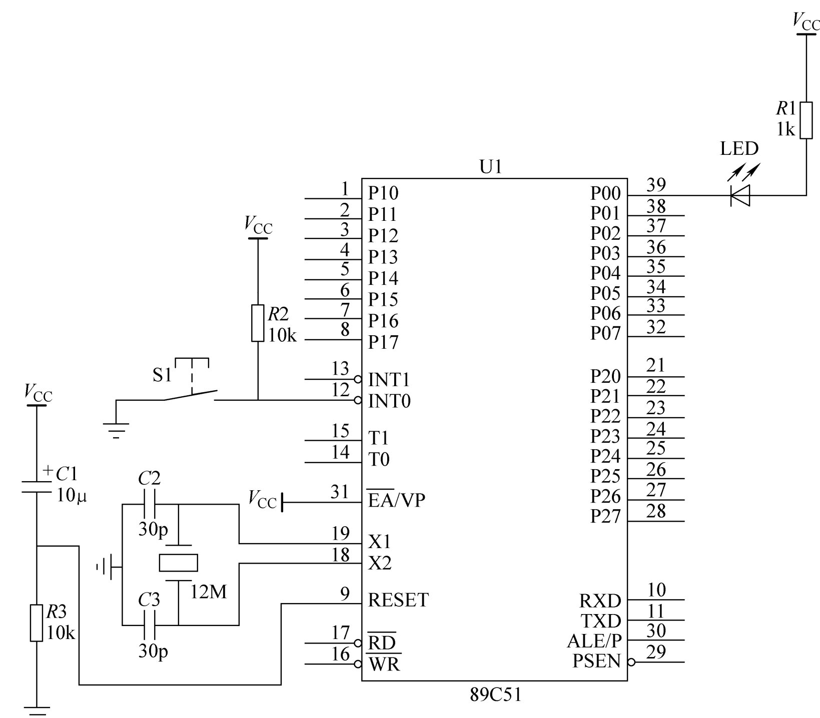


图 2-15　硬件原理

程序设计：

```
#include <reg52.h>
#include <absacc.h>
#define uchar unsigned char

sbit LED = P1^0;
void delays (void)
{
uchar i;
for (i = 255; i > 0; i - -);
}

void main (void)
{
P1 = 0xff;                              // 初始化端口
EA = 1; IT0 = 1; EX0 = 1;               // 初始化外中断标志位
while (1)
```

```
{
        delays ( );                    // 等待中断
}
}

/* 外中断 0 的中断服务子程序 */
void intersvr0 (void) interrupt 0 using 1
{
LED = ! LED;                           // LED 状态取反
}
```

2.7 串行通信

单片机除了需要控制外围器件完成特定的功能外，在很多应用中还要完成单片机和单片机之间、单片机和外围器件之间，以及单片机和计算机之间的数据交换和指令的传输，称为单片机的通信。单片机的通信方式可以分为并行通信和串行通信。并行方式传送 1 个字节的数据至少需要 8 条数据线。一般来讲 MCS-51 单片机与打印机等外围设备连接时，除 8 条数据线外，还要状态、应答等控制线，当传送距离过远时电线要求过多，成本会增加很多。单片机的串行通信方法较为多样，传统的串行通信方式是通过单片机自带的串行口进行 RS232 方式的通信。串行通信是以一位数据线传送数据的位信号，即使加上几条通信联络控制线，也比并行通信用线少。因此，串行通信适合远距离数据传送，如大型主机与其远程终端之间，处于两地的计算机之间，采用串行通信就非常经济。

2.7.1 串行通信概述

1. 并行通信与串行通信

所谓通信就是指计算机与计算机或外围设备之间的数据传送。通信的数据是由数字“0”和“1”构成的具有一定规则并反映确定信息的一个数据或一批数据。这种数据传送有两种基本方式，即并行通信和串行通信。

并行通信比较简单，根据 CPU 字长和总线特点以及外围设备数据口的宽度，可分为不同位数的并行通信，如 16 位并行通信、32 位并行通信等。并行通信的特点是数据的每位被同时传输出去或接收进来，传输速度较快。与并行通信不同，串行通信其数据传输是逐位传输的，因而相同条件下，比并行通信传输速度慢。

虽然串行通信速度比并行通信慢，但采用串行通信，不管发送或接收的数据是多少，最多只需要两根线，一根用于发送，另一根用于接收。根据串行通信的不同操作方式，还可以将发送接收线合二为一，成为发送/接收复用线（如半双工）。即使在实际应用中可能还要加一些信号线，但在多字节数据通信中，串行通信与并行通信相比，其工程造价要低得多。因此，串行通信已被越来越广泛地采用。

2. 异步通信和同步通信

1）异步通信：异步通信传输的数据格式一般由 1 个起始位、7 个或 8 个数据位、1～2

个停止位和一个校验位组成。它用一个起始位表示字符的开始，用停止位表示字符的结束。其每帧的格式如图 2-16 所示。

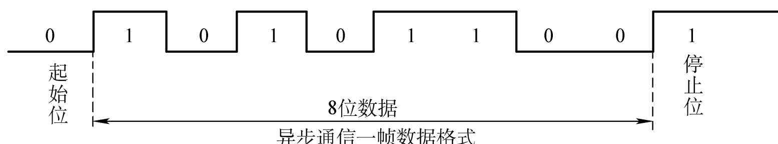


图 2-16　异步通信

在一帧格式中，先是一个起始位 0，然后是 8 个数据位，规定低位在前，高位在后，接下来是奇偶校验位（可以省略)，最后是停止位 1。用这种格式表示字符，则字符可以一个接一个地传送。

在异步通信中，通信双方采用独立的时钟，起始位触发双方同步时钟。在异步通信中 CPU 与外设之间必须有两项规定，即字符格式和波特率。字符格式的规定是双方能够在对同一种 0 和 1 的串理解成同一种意义。原则上字符格式可以由通信的双方自由制定，但从通用、方便的角度出发，一般还是使用一些标准为好，如采用 ASCII 标准。

2）同步通信：在同步通信中所传输的数据格式是由多个数据组成，每帧有一个或两个同步字符作为起始位以触发同步时钟开始发送或接收。在异步通信中，每个字符要用起始位和停止位作为字符开始和结束的标志，占用了时间，所以在数据块传递时，为了提高速度，常去掉这些标志，采用同步传送。由于数据块传递开始要用同步字符来指示，同时要求由时钟来实现发送端与接收端之间的同步，故硬件较复杂。同步传输方式比异步传输方式速度快，这是它的优势。但同步传输方式也有其缺点，即它必须要用一个时钟来协调收发器的工作，所以它的设备也较复杂。同步通信数据格式如图 2-17 所示。

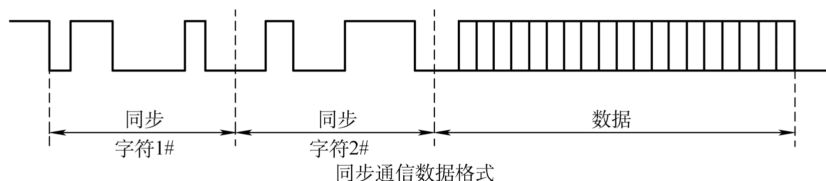


图 2-17　同步通信

3. 串行通信波特率

波特率是串行通信中的一个重要概念。波特率（bitpersecond）是单位时间里传输的数据位数，即：波特率 = 1bit/s。例如，数据传送的速率是 240 字符/s，而每个字符如上述规定包含 10 数位，则传送波特率为 2400 波特。波特率的倒数就是传送每位数据所需要的时间。相互通信的双方必须具有相同的波特率，否则无法成功地完成数据通信。异步通信一般速度较慢，一般适用于 50 ~ 9600bit/s。

4. 串行通信操作模式

在异步方式和同步方式中，串行通信都具有多种操作模式，常用于数据通信的传输方式有单工、半双工、全双工和多工方式。

单工方式：双方通信数据仅按一个固定方向传送，系统定型后也就固定了发送方和接收方。因而这种传输方式的用途有限，常用于串行口的打印数据传输与简单系统间的数据采

集。

半双工方式：通信双方都具有收发器，数据可实现双向传送，但不能同时进行，实际的应用采用某种协议实现收/发开关转换。

全双工方式：通信双方都具有收发器，允许双方同时进行数据双向传送，但一般全双工传输方式的线路和设备较复杂。

多工方式：以上三种传输方式都是用同一线路传输一种频率信号，为了充分地利用线路资源，可通过使用多路复用器或多路集线器，采用频分、时分或码分复用技术，即可实现在同一线路上资源共享功能，我们称之为多工传输方式。

2.7.2 MCS-51 单片机的串行接口结构

对于单片机来说，为了进行串行数据通信，同样也需要有相应的串行接口电路。只不过这个接口电路不是单独的芯片，而是集成在单片机内部，成为单片机芯片的一个组成部分。MCS-51 单片机内部有一个可编程的全双工的串行通信口，它可用作异步通信方式（UART），与串行传送信息的外部设备相连接，或用于通过标准异步通信协议进行全双工的 8051 多机系统，也可以通过同步方式，使用 TTL 或 CMOS 移位寄存器来扩充 I/O 口。

MCS-51 单片机内的 SBUF 是串行口缓冲寄存器，包括发送寄存器和接收寄存器。它们有相同的名字和地址空间，但不会出现冲突，因为一个只能被 CPU 读出数据，一个只能被 CPU 写入数据。MCS-51 单片机通过引脚 RXD（P3.0，串行数据接收端）和引脚 TXD（P3.1，串行数据发送端）与外界通信。这个通信口既可以用于网络通信，亦可实现串行异步通信，还可以构成同步移位寄存器使用。如果在串行口的输入/输出引脚加上电平转换器，就可方便地构成标准的 RS232 接口。

2.7.3 MCS-51 的串行口数据缓冲器 SBUF

MCS-51 单片机串行口寄存器结构如图 2-18 所示。SBUF 为串行口的收发缓冲器，它是一个可寻址的专用寄存器，其中包含了接收器和发送器寄存器，可以实现全双工通信。MCS-51 的串行数据传输很简单，只要向发送缓冲器写入数据即可发送数据。而从接收缓冲器读出数据即可接收数据。从图 2-18 中可看出，接收缓冲器前还加上一级输入移位寄存器，MCS-51 的这种结构目的在于接收数据时避免发生数据帧重叠现象，以免出错。而发送数据时就不需要这样设置，因为发送时，CPU 是主动的，不可能出现这种现象。

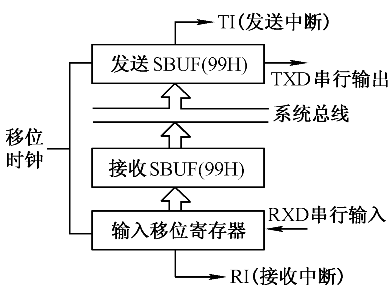


图 2-18　MCS-51 的串行口数据缓冲器 SBUF

2.7.4 串行通信控制寄存器

1. 串行中断控制寄存器 SCON

在中断系统中我们已经分析了 SCON 控制寄存器，它是一个可寻址的专用寄存器，用于串行数据的通信控制，单元地址是 98H，其结构格式见表 2-9。

下面我们对各控制位功能介绍如下：

SM0、SM1：串行口工作方式控制位见表 2-10。

表 2-9　SCON 寄存器结构

SCON 寄存器结构								
SCON	D7	D6	D5	D4	D3	D2	D1	D0
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
位地址	9FH	9EH	8DH	9CH	9BH	9AH	99H	98H

表 2-10　SM0、SM1 位置位作用

SM0、SM1	工作方式	功能描述	波特率
0 0	方式 0	8 位移位寄存器	$f_{osc}/12$
0 1	方式 1	8 位 UART	可变
1 0	方式 2	9 位 UART	$f_{osc}/64$ 或 $f_{osc}/32$
1 1	方式 3	9 位 UART	可变

表中 f_{osc} 为晶振频率。

SM2：多机通信控制位。多机通信工作于方式 2 和方式 3，所以 SM2 位主要用于方式 2 和方式 3。接收状态：当串行口工作于方式 2 或 3，以及 SM2 = 1 时，只有当接收到第 9 位数据（RB8）为 1 时，才把接收到的前 8 位数据送入 SBUF，且置位 RI 发出中断申请，否则会将接受到的数据放弃。当 SM2 = 0 时，就不管第 9 位数据是 0 还是 1，都把前 8 位数据送入 SBUF，并发出中断申请。工作于方式 0 时，SM2 必须为 0。

REN：接收允许控制位。REN 用于控制数据接收的允许和禁止，REN = 1 时，允许接收；REN = 0 时，禁止接收。由软件置位以允许接收，又由软件清零来禁止接收。

TB8：要发送数据的第 9 位。在方式 2 或方式 3 中，要发送的第 9 位数据，根据需要由软件置 1 或清零。例如，可约定作为奇偶校验位，或在多机通信中作为区别地址帧或数据帧的标志位。在方式 2 和方式 3 中，TB8 是要发送的第 9 位数据位。在多机通信中同样亦要传输这一位，并且它代表传输的地址或是数据，TB8 = 0 时为数据，TB8 = 1 时为地址。

RB8：接收到的数据的第 9 位。在方式 0 中不使用 RB8。在方式 1 中，若 SM2 = 0，RB8 为接收到的停止位。在方式 2 或方式 3 中，RB8 为接收到的第 9 位数据。在方式 2 和方式 3 中，RB8 存放接收到的第 9 位数据，用以识别接收到的数据特征。

TI：发送中断标志。在方式 0 中，第 8 位发送结束时，由硬件置位。在其他方式的发送停止位前，由硬件置位。TI 置位既表示一帧信息发送结束，同时也是申请中断，可根据需要，用软件查询的方法获得数据已发送完毕的信息，或用中断的方式来发送下一个数据。TI 必须用软件清零。可寻址标志位。

RI：接收中断标志位。在方式 0 中，当接收完第 8 位数据后，由硬件置位。在其他方式中，在接收到停止位的中间时刻由硬件置位（例外情况见关于 SM2 的说明）。RI 置位表示一帧数据接收完毕，可用查询的方法获知或者用中断的方法获知。RI 也必须用软件清零。可寻址标志位。

2. 电源控制寄存器 PCON

PCON 主要是为 CHMOS 型单片机的电源控制而设置的专用寄存器，单元地址是 87H，其结构格式见表 2-11。

表 2-11　电源控制寄存器 PCON

PCON 电源管理寄存器结构								
PCON	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SMOD	—	—	—	GF1	GF0	PD	IDL

在 CHMOS 型单片机中，除 SMOD 位外，其他位均为虚设的。SMOD 是串行口波特率倍增位，当 SMOD = 1 时，串行口波特率加倍。系统复位默认为 SMOD = 0。

3. 中断允许寄存器 IE

中断允许寄存器在中断系统中已经阐述，这里重述一下对串行口有影响的位 ES。ES 为串行中断允许控制位，见表 2-12。ES = 1，允许串行中断，ES = 0，禁止串行中断。

表 2-12　IE 中断允许控制寄存器结构

IE 中断允许控制寄存器结构								
位符号	EA	—	—	ES	ET1	EX1	ET0	EX0
位地址	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H

4. 串行通信工作方式

8051 单片机的全双工串行口可编程为 4 种工作方式，现分述如下：

（1）方式 0 为移位寄存器输入/输出方式。可外接移位寄存器以扩展 I/O 口，也可以外接同步输入/输出设备。8 位串行数据都是从引脚 RXD 输入或输出，引脚 TXD 用来输出同步脉冲。

1）数据发送：串行数据从引脚 RXD 输出，引脚 TXD 输出移位脉冲。CPU 将数据写入发送寄存器时，立即启动发送，将 8 位数据以 $f_{osc}/12$ 的固定波特率从 RXD 输出，低位在前，高位在后。发送完一帧数据后，发送中断标志 TI 由硬件置位。使用方式 0 实现数据的输入输出时，实际上是把串行口变成并行口用。

2）数据接收：当串行口以方式 0 接收时，先置位允许接收控制位 REN。此时，引脚 RXD 为串行数据输入端，引脚 TXD 仍为同步脉冲移位输出端。当 RI = 0 和 REN = 1 同时满足时，开始接收。当接收到第 8 位数据时，将数据移入接收寄存器，并由硬件置位 RI。

3）波特率：方式 0 的波特率固定为主振频率的 1/12。若 f_{osc} = 6MHz，则波特率为 500bit/s，即 2μs 移位一次。

（2）方式 1 为波特率可变的 10 位异步通信接口方式。发送或接收一帧信息，包括 1 个起始位 0，8 个数据位和 1 个停止位 1。

1）数据发送：当 CPU 执行一条指令将数据写入发送缓冲 SBUF 时，就启动发送。在串行口由硬件自动加入起始位和停止位，构成一个完整的帧格式，然后在移位脉冲的作用下串行数据从引脚 TXD 输出。发送完一帧数据后引脚 TXD 一直维持在“1”状态下，并由硬件置位 TI 通知 CPU 可以发送下一个字符。

2）数据接收：在 REN = 1 时，串行口采样引脚 RXD，当采样到 1 至 0 的跳变时，确认是开始位 0，就开始接收一帧数据。只有当 RI = 0 且停止位为 1 或者 SM2 = 0 时，停止位才进入 RB8，8 位数据才能进入接收寄存器，并由硬件置位中断标志 RI；否则信息丢失。所以在方式 1 接收时，应先用软件清零 RI 和 SM2 标志。

3）波特率：方式 1 的波特率是可变的，由定时/计数器 1 的计数溢出率来决定，公式为

波特率 $=2^{SMOD}$（定时器1溢出率）/32

4）SMOD：PCON寄存器的最高位值，SMOD=1表示波特率增倍。

（3）方式2为固定波特率的11位UART方式。即1个起始位，9个数据位和1个停止位。在方式2中，字符还是8个数据位，第9个数据位既可以作为奇偶校验位也可作为控制位使用，功能由用户自己定。

1）数据发送：发送的串行数据由引脚TXD输出，一帧信息为11位，附加的第9位来自SCON寄存器的TB8位，用软件置位或复位。准备好第9个数据位之后，当CPU执行一条数据写入SUBF的指令时，就启动发送器发送。发送一帧信息后，置位中断标志TI，其过程与方式1相同。

2）数据接收：方式2的接收过程也与方式1基本相同，不同之处在于第9个数据位，串行口把收到的8位数据送入SBUF，把收到的第9位数据送入RB8。

3）波特率：方式2的波特率由PCON中的选择位SMOD来决定，可由下式表示

波特率 $=2^{SMOD}f_{osc}/64$，也就是当SMOD=1时，波特率为 $f_{osc}/32$，当SMOD=0时，波特率为 $f_{osc}/64$。

（4）方式3为波特率可变的11位UART方式。除波特率外，其余与方式2相同。

2.7.5 波特率选择与设置

如前所述，在串行通信中，收发双方的数据传送率（波特率）要有一定的约定。在8051串行口的4种工作方式中，方式0和2的波特率是固定的，而方式1和3的波特率是可变的，由定时器T1的溢出率控制。各种方式的波特率如下：

方式0：波特率 $=f_{osc}/12$

方式2：波特率 $=2^{SMOD}f_{osc}/64$

方式1和方式3（定时器T1作为波特率发生器）：波特率 $=2^{SMOD}$（定时器1溢出率）/32

T1溢出率 = T1计数率/产生溢出所需的周期数

式中，T1计数率取决于它工作在定时器状态还是计数器状态。当工作于定时器状态时，T1计数率为 $f_{osc}/12$；当工作于计数器状态时，T1计数率为外部输入频率，此频率应小于 $f_{osc}/24$。产生溢出所需周期与定时器T1的工作方式、T1的预置值有关。

定时器T1工作于方式0：溢出所需周期数 $=8192-X$，X 为初值。

定时器T1工作于方式1：溢出所需周期数 $=65536-X$，X 为初值。

定时器T1工作于方式2：溢出所需周期数 $=256-X$，X 为初值。

因为方式2为自动重装入初值的8位定时/计数器模式，所以用它来做波特率发生器最恰当。

当时钟频率选用11.0592MHz时，极易获得标准的波特率，所以很多单片机系统选用这个看起来“怪”的晶振就是这个道理。方式1和方式3的常用波特率的设置见表2-13。

有了上述表格，在C语言中可以很方便地对串口进行初始化。如在11.0592MHz晶振下，设置串行口用9.6K的传输率进行传输，工作于工作方式3，用C语言可进行如下设定：

```
TMOD=0X20;          //定时器工作方式2，初值自动装入
TL1=0XFD;           //定时器初值低位
```

```
TH1 = 0XFD;            //定时器初值高位
SCON = 0XD8;           //串行工作方式 3
PCON = 0X00;           //波特率不增倍
TR1 = 1;               //启动定时器 1
```

表 2-13　数据传输率与晶振频率的关系

数据传输率	f_{osc} = 6MHz			f_{osc} = 12MHz			f_{osc} = 11.0592MHz		
	SMOD	TMOD	TH1	SMOD	TMOD	TH1	SMOD	TMOD	TH1
62.5K				1	20	FFH			
19.2K							1	20	FDH
9.6K							0	20	FDH
4.8K				1	20	F3H	0	20	FAH
2.4K	1	20	F3H	0	20	F3H	0	20	F4H
1.2K	1	20	E6H	0	20	E6H	0	20	E8H
600	1	20	CCH	0	20	CCH	0	20	D0H

2.7.6　RS232 标准接口总线及串行通信设计

通过以上枯燥的理论学习之后对 MCS-51 单片机的串行口有了基本了解，接下来重点介绍单片机与 PC 之间的通信。相信每个初学者都会对这部分内容感兴趣。

在工业自动控制、智能产品中，单片机应用越来越广泛，同时也需要对数据进行较复杂的处理。由于单片机的运算能力较差，在处理复杂数据时速度较慢，所以需要借助计算机进行运算。因此，单片机与 PC 间的通信便显得非常重要。

大多数的计算机都具有 RS232C 接口，尽管它的性能指标并非很好。在广泛的市场支持下依然常盛不衰。就使用而言，RS232 也确实有其优势：仅需 3 根线便可在两个数字设备之间全双工的传送数据。不过，RS232C 的控制要比使用并行通信的打印机接口更难于控制。RS232C 使用了远比并行口更多的寄存器。这些寄存器用来实现串行数据的传送及 RS232C 设备之间的握手与流量控制。PC 上的 RS232 口各引脚功能如图 2-19 所示。

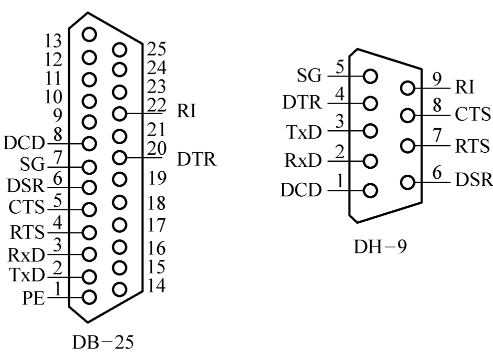


图 2-19　RS232 接口定义

1. RS232 总线标准

串行通信接口标准以 RS232C 为主。RS232C 标准是美国 EIA 与 BELL 等公司一起开发的，它适合于数据传输率在 0 ~ 20000bit/s 范围内的通信。RS232C 还对电器特性、逻辑电平和各种信号线功能都作了规定。RS232C 使用 −3 ~ −25V 表示数字“1”，使用 3 ~ 25V 表示数字“0”，RS232C 在空闲时处于逻辑“1”状态。在开始传送时，首先产生一个起始位，起始位为一个宽度的逻辑“0”，紧随其后的为要传送的数据，所要传送的数据由最低位开始送出，最后以一个结束位标志表示该字节传送完毕，结束位为一个宽度的逻辑“1”。

2. RS232C 接口电路

由于 RS232C 信号与 MCS-51 单片机信号电平不一致（前者为 RS232 电平，后者为 TTL 电平），因此，采用 RS232C 与单片机通信时必须要进行信号电平转换。目前，RS232C 与

TTL 电平转换最常用的芯片有 MAX232、MC1488 等，以下就以 MAX232 为例，介绍其接口电路。MAX232 芯片是 MAXIM 公司生产的、包含两路接收器和驱动器的 IC 芯片，外部引脚和内部电路如图 2-20 所示。

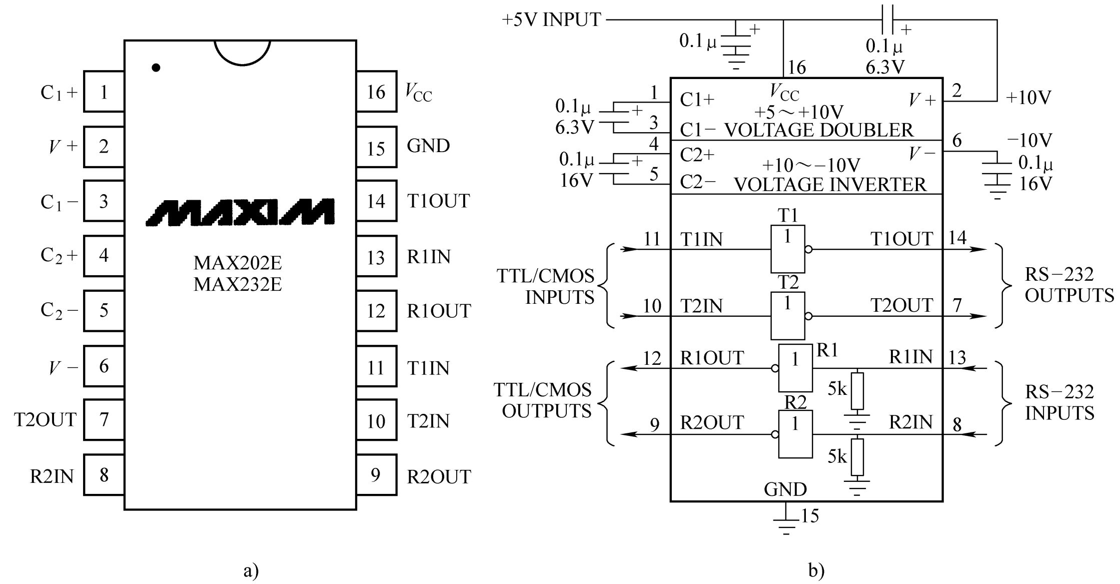


图 2-20　MAX232 芯片接口定义

a）引脚排列　b）内部功能

实际应用中可以接成如图 2-21 所示电路，实现单片机与 PC 间数据通信。

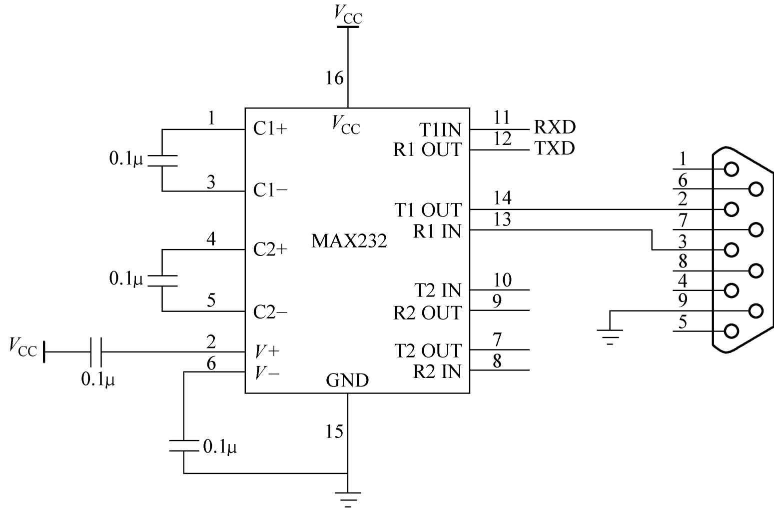


图 2-21　MAX232 串行通信电路

MAX232 内部有两组收、发器，实际应用中可以从中任选一组使用，电平转换的硬件电路在此不作更多介绍，读者可以从网上找到许多类似资料。

3. 单片机与 PC 通信的软硬件设计

51 单片机有一个全双工的串行通信口，所以单片机和计算机之间可以方便地进行串行

通信。进行串行通信时要满足一定的条件，比如计算机的串口是 RS232 电平的，而单片机的串口是 TTL 电平的，两者之间必须有一个电平转换电路，我们采用了专用芯片 MAX232 进行转换，虽然也可以用几个晶体管进行模拟转换，但还是用专用芯片更简单可靠。具体硬件电路如图 2-21，这是最简单的连接方法，但是对我们来说已经足够使用了。

为了能够在计算机端看到单片机发出的数据，我们必须借助一个 Windows 软件进行观察，这里我们推荐一个免费的计算机串口调试软件——串口调试助手。此软件如图 2-22 所示，可设定串口号、波特率、校验位等参数，非常实用。



图 2-22　串口调试软件界面

程序设计：利用单片机串行口将 PC 发送的数据信号接收并返回给 PC，利用串口调试助手发送并验证收到的数据是否正确。在使用中注意波特率两部分一定要统一，否则数据将出错。单片机上程序如下：

```
/*****************************/
/*        杭州电子 & 计算机工作室        */
/*        http://www.hificat.com        */
/*        串行口演示程序        */
/*        目标器件：AT89C51        */
/*        晶振：11.0592MHz        */
/*        编译环境：Keil 7.50A        */
/*****************************/
/*******************包含头文件*******************/
#include “reg51.h"
#include <absacc.h>
```

```
/ * * * * * * * * * * * * * * * * * * * *共阳 LED 段码表* * * * * * * * * * * * * * * * * * * */
unsigned char code tab [ ] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80,
0x90};
/ * * * * * * * * * * * * * * * * * * * * *定义全局变量* * * * * * * * * * * * * * * * * * * */
unsigned char dat;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能：串行口初始化程序
入口参数：
出口参数：
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void Init_Com (void)
{
    TMOD = 0x20;
    PCON = 0x00;
    SCON = 0x50;
    TH1 = 0xFd;
    TL1 = 0xFd;
    TR1 = 1;
}
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能：LED 数码管延时程序
入口参数：
出口参数：
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void delay (void)
{
int k;
for (k=0; k<600; k++);
}
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能：LED 数码管显示程序
入口参数：k
出口参数：
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void display (int k)
{
    P2=0xfe;                                   //位选
    P0=tab [k/1000];                           //显示千位数字
    delay ();                                  //延时
```

```
    P2 = 0xfd;                              //位选
    P0 = tab [k%1000/100];                  //显示百位数字
    delay ();                               //延时
    P2 = 0xfb;                              //位选
    P0 = tab [k%100/10];                    //显示十位数字
    delay ();                               //延时
    P2 = 0xf7;                              //位选
    P0 = tab [k%10];                        //显示个位数字
    delay ();                               //延时
    P2 = 0xff;                              //位选
}
/*********************************************
函数功能：主程序
入口参数：
出口参数：
*********************************************/
void main ()
{
    P2 = 0xff;
    P0 = 0xff;
    EA = 1;
    ES = 1;
    Init_Com ();
    while (1)
      {
      display (dat);                        //数据显示
      }
}
/*********************************************
函数功能：串行中断服务程序
入口参数：
出口参数：
*********************************************/
serial () interrupt 4 using 1
{
    if (RI) RI = 0;
    dat = SBUF;
}
```

第 3 章　C 语言数据类型、运算符、表达式

3.1　C 语言概论

3.1.1　C 语言的发展过程

C 语言是在 1969 年～1973 年这段时间问世的，后来到了 1978 年，由美国电话电报公司（AT&T）贝尔实验室正式发表了 C 语言，同时由 B. W. Kernighan 和 D. M. Ritchit 合著了著名的"THE C PROGRAMMING LANGUAGE"一书，通常简称为《K&R》或"白皮书"。但是，在"白皮书"中并没有定义一个完整的标准 C 语言，到 1983 年，由美国国家标准学会在此基础上制定了一个 C 语言标准，我们称之为 ANSI C。到了今天，C 语言已经成为了一门在整个计算机领域的普遍应用的语言了。

3.1.2　C 语言的特点

C 语言是一种结构化语言。它层次清晰，便于按模块化方式组织程序，易于调试和维护，语言见解紧凑，使用方便、灵活。它不仅具有丰富的运算符和数据类型，以便于实现各类复杂的数据结构，它还可以直接访问内存地址，能进行位（bit）操作，尤其能够胜任开发操作系统的工作。由于 C 语言实现了对硬件的编程操作，因此 C 语言既有高级语言的功能，也有低级语言的优势。它可用于系统软件的开发，同样也适用于应用软件的开发。此外，C 语言还具有效率高，可移植性强等特点。如果原来使用的是汇编语言编写的程序，在若干时间后再去做升级和维护就会感觉非常不便，别人写的程序不宜被读懂，但在维护 C 语言写的程序时，此时其优势就大大体现出来了。

3.1.3　C 源程序的结构特点

为了说明 C 语言的结构特点，我们先来看以下两个典型的程序例子。这两个程序由简到难，但从中体现了 C 语言在组成结构上的特点。从这些例子中，我们可以了解到组成一个 C 源程序的基本部分和书写格式。

【例 3-1】

```
main（）
{
printf（"朋友，您好！\n"）；
}
```

main 是主函数的函数名，表示这是一个主函数。每一个 C 程序都必须有，而且只能有

一个主函数（main 函数）。函数调用语句——printf 函数的功能是在显示器上显示我们要输出的内容。printf 函数是一个由系统定义好的标准函数，在程序中我们可以直接调用。

【例 3-2】

```
int min (int a, int b); /* 函数说明 */
main () { /* 主函数 */
int x, y, z; /* 变量说明 */
printf ("input two numbers: \n"); scanf ("%d%d", &x, &y); /* 输入 x, y 值 */
z = min (x, y); /* 调用 max 函数 */
printf ("min num = %d", z); /* 输出 */
}
int min (int a, int b) { /* 定义 max 函数 */
if (a < b) return a; else return b; /* 把结果返回主调函数 */
}
```

此函数的功能是由用户输入两个整数，然后输出其中一个较小的数。

例 3-2 程序的功能是由用户输入两个整数，程序执行后输出其中较小的数。本程序由两个函数组成，一个主函数，一个 min 函数。函数之间是并列的关系。在主函数中可以调用其他函数。min 函数的功能是比较两个数，然后把较小的数返回给主函数。min 函数是一个用户自定义函数。因此在主函数中要给出说明（程序的第一行）。由此可见，在程序的说明部分中，不仅可以有变量的说明，还可以有函数的说明。关于函数的详细内容将在第五章介绍。在例程中，可以看到程序每行后面有用/＊和＊/括起来的内容，我们称之为注释部分，程序在编译时，不会执行这部分内容。

例 3-2 程序执行的过程是，首先出现让用户输入两个数的提示信息，scanf 函数的作用是将这两个数送入变量 x，y 中，然后调用 min 函数，并把 x，y 的值传送给 min 函数的形式参数 a，b。min 函数中的语句用来比较变量 a，b 数值的大小，把小的那个数返回给主函数的变量 z，最后再显示输出 z 的值。

下面总结一下 C 语言的特点：

1）一个 C 语言源程序可以由一个或多个源文件组成。

2）每个源文件可由一个或多个函数组成。

3）一个源程序不论由多少个文件组成，都有一个且只能有一个 main 函数，即主函数。

4）源程序中可以有预处理命令（include 命令仅为其中的一种），预处理命令通常应放在源文件或源程序的最前面。

5）每一个说明，每一个语句都必须以分号结尾。但预处理命令，函数头和花括号“}”之后不能加分号。

6）标识符，关键字之间必须至少加一个空格以示间隔。若已有明显的间隔符，也可不再加空格来间隔。

3.1.4 C 语言的字符集

字符是组成语言的最基本的元素。C 语言字符集由字母，数字，空格，标点和特殊字符

组成。在字符常量，字符串常量和注释中还可以使用汉字或其他可表示的图形符号。

1. 字母

小写字母 a ~ z 共 26 个；

大写字母 A ~ Z 共 26 个；

2. 数字

0 ~ 9 共 10 个。

3. 空白符

空格符、制表符、换行符等统称为空白符。空白符只在字符常量和字符串常量中起作用。在其他地方出现时，只起间隔作用，编译程序对它们忽略不计。因此在程序中使用空白符与否，对程序的编译不发生影响，但在程序中适当的地方使用空白符将增加程序的清晰性和可读性。

4. 标点和特殊字符

3.1.5 C 语言词汇

在 C 语言中使用的词汇分为六类：标识符，关键字，运算符，分隔符，常量，注释符等。

1. 标识符

在程序中使用的变量名、函数名、标号等统称为标识符。除库函数的函数名由系统定义外，其余都由用户自定义。C 语言规定，标识符只能是字母（A ~ Z，a ~ z）、数字（0 ~ 9）、下划线（）组成的字符串，并且其第一个字符必须是字母或下划线。

以下标识符是合法的：

a，BOOKS ，abc5

以下标识符是非法的：

4s 以数字开头

s#T 出现非法字符#

在使用标识符时还必须注意以下几点：

（1）C 语言编译器版本的不同，使得标识符的长度也会有所不同。例如有些版本中规定标识符前八位有效。

（2）在标识符中，大小写是严格区分的。例如 TEACHER 和 teacher 是两个不同的标识符。

（3）定义标识符时，取的名字应尽量有直观的意义，以便于阅读理解，做到“顾名思义”。

2. 关键字

关键字是在 C 语言系统中具有特定意义的字符串，通常也称为保留字。用户定义的标识符名字不能与关键字相同。C 语言的关键字分为以下几类：

（1）类型说明符：用来定义变量、函数或其他数据结构的类型。如例程中用到的 int，double 等。

（2）语句定义符：用来表示一个语句的功能意义。如我们后面要讲到的“if else”就是条件语句的语句定义符。

（3）预处理命令字：表示预处理命令的关键字。如例程中用到的“include”。

3. 运算符

C 语言中含有相当丰富的运算符。运算符是告诉编译程序执行特定算术或逻辑操作的符号。C 语言有三大运算符：算术、关系与逻辑、位操作。另外，C 语言还有一些特殊的运算符，用于完成一些特殊的任务。

4. 分隔符

在 C 语言中的分隔符有逗号和空格两种。逗号主要用在数据类型说明和函数参数表中，分隔各个变量；而空格多用于语句各单词之间，做间隔符。

5. 常量

C 语言中使用的常量可分为数字常量、字符常量、字符串常量、符号常量、转义字符等多种。

6. 注释符

C 语言的注释符是从“／＊”开头到以“＊／”结尾的内容，在“／＊”和“＊／”之间的内容即为注释。程序在编译时，不对这些注释内容做任何处理。注释可出现在程序中的任何位置，用来向用户提示或解释程序的意义，方便了程序的编写及调试、维护工作。

3.2　数据类型、运算符与表达式

3.2.1　C 语言的数据类型

计算机中的程序，离不开数据个这个单元，它是计算机操作的对象，我们将数据的不同格式叫做数据类型；而数据结构则是按一定的数据类型进行一些组合和构架。

在 C 语言中，数据类型可分为：基本数据类型，构造数据类型，指针类型，空类型四大类，如图 3-1 所示。

下面我们来看一下这 4 种数据类型的特点：

（1）基本数据类型：它的数据不可以再进行分解。

（2）构造数据类型：根据已定义的一个或多个数据类型用构造的方法来定义的。也就是说，一个构造类型的值可以由若干个“成员”或“元素”组成。其“成员”可以是一个基本数据类型或构造类型。在 C 语言中，构造类型有以下几种：

1）数组类型

2）结构体类型

3）共用体类型

（3）指针类型：指针是一个特殊的变量，它里面存储的数值被解释成为内存里的一个地址，也是 C 语言的精华所在。虽然指针变量的值类似于整型量，但从其数据类型意义来看，这是两种完全不同的类型，所以不能混为一谈。

（4）空类型：函数在被调用完成后，通常会返回一个函数值。函数值有一定的数据类型的，应在函数定义及函数说明中

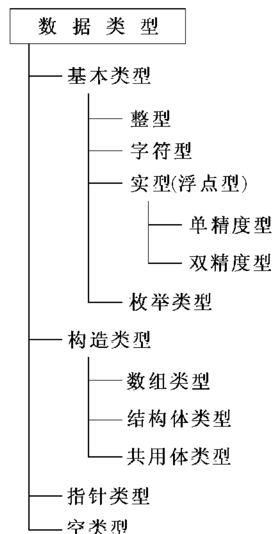


图 3-1　数据类型

加以说明，如在例 3-2 中给出的 min 函数定义中，函数头为：int min（int a，int b）；其中“int”表示 min 函数的返回值为整型数据类型。但是，我们也经常会碰到一些函数，调用后并不需要向调用者返回函数值，这时，我们应该如何来书写呢？此时，我们将这种函数定义为“空类型”，其类型说明符为 void。如将 int min（int a，int b）；改为 void min（int a，int b）；即可。

在本节中，我们先介绍基本数据类型中的整型、浮点型和字符型。其余类型在以后各节中陆续介绍。

3.2.1.1 常量与变量

对于基本数据类型量，根据变量值在程序执行过程中是否发生变化，又可分为常量和变量两种。在程序执行过程中，其值不发生改变的量称为常量，其值可变的量称为变量。每个变量都会有个名字，并在内存中占据一定的存储单元，所占存储单元的数量根据数据类型的不同而不同。在程序中，常量是可以不经说明而直接引用的，而变量则必须先定义后使用。

1. 常量和符号常量

常量与变量相对应，在程序执行的过程中，其值不能发生改变的量称为常量。常量与变量不同，可以有不同的数据类型，如：1，3，5 为整型常量；6.9，－1.85 为实型常量；‘c’，‘d’为字符常量。常量可以用一个标识符来说明，符号常量在使用之前必须先定义，其一般形式为

#define　标识符　常量

其中，#define 是一条编译预处理命令（预处理命令都以“#”开头），称为宏定义命令，其功能是把该标识符定义为其后的常量值。一经定义，凡在程序中所有出现该标识符的地方均用之前定义好的常量来代替。习惯上符号常量的标识符用大写字母来表示，变量标识符用小写字母，以示区别。

如下面的程序所示：

【例 3-3】

```
#define PI 3.14
main（）
{
float r，s；
r =5.3；
s = PI * r * r；
printf（“半径为 r 的圆面积为%f”，s）；
}
```

程序的第一句话“#define PI 3.14”定义了一个符号常量 PI，它的值为圆周率 3.14，由此一来，在后面的程序代码中，凡是出现 PI 的地方，都代表 3.14 这个数。

使用符号常量的好处是：意义清楚；修改参数非常方便。如程序中很多地方都要用到这

个变量，而数值又需要经常做改动，这样的情况下使用符号常量就可以做到“一改全改，一次改成”。

2. 变量

在程序运行中，其值可以改变的量称为变量。一个变量有一个名字，在内存中占据一定的存储单元，如图 3-2 所示。变量必须先定义再使用，一般放在函数体的开头部分，当然，如果是全局变量的话，就放在函数体外面了。

图 3-2　变量与存储单元

3.2.1.2　整型数据

1. 整型常量的表示方法

整型常量就是整型的常数。在 C 语言中，整型常量可以分为八进制、十进制和十六进制三种。

1）十进制数没有前缀。用数码 0 ~ 9 来表示。

如以下各数是合法的十进制整型常数：

257、 -518、55535、1968；

在程序中是根据前缀来区分各种进制数的。因此在书写常数时不要把前缀弄错造成结果不正确。

2）八进制数：必须以 0 开头，即以 0 作为八进制数的前缀。用数码 0 ~ 7 来表示。八进制数通常是无符号数。

以下各数是合法的八进制数：

016（相当于十进制数 14）、0110（相当于十进制数 72）、0177776（相当于十进制数 65534）；

以下各数不是合法的八进制数：

156（无前缀 0）、01B2（包含了非八进制数字）、 -0170（不应该有负号）。

3）十六进制数：以 0X 或 0x 开头。其数码取值为 0 ~ 9，A ~ F 或 a ~ f。

以下各数是合法的十六进制整常数：

0X2B（相当于十进制 43）、0XB0（相当于十进制 176）、0XFFFF（相当于十进制 65535）；

以下各数不是合法的十六进制整常数：

7B（无前缀 0X）、0X5H（含有非十六进制数码）。

2. 整型变量

（1） 整型变量的分类

1）基本型：类型说明符为 int，在内存中占 2 个字节。

2）无符号型：类型说明符为 unsigned。无符号型又可为 unsigned int 或 unsigned。无符号类型量所占的内存空间字节数与相应的有符号类型量相同，但由于省去了符号位，故不能表示负数，也因此表示正数的数值范围有所扩大。

有符号整型变量：最大表示 32767

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

无符号整型变量：最大表示 65535

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

KEIL uVision2 C51 编译器所支持的数据类型见表 3-1。

表 3-1　数据类型表

数据类型	长度/bit	长度/Byte	值　域
unsigned char	8	1	0 ~ 255
signed char	8	1	−128 ~ +127
unsigned int	16	2	0 ~ 65535
signed int	16	2	−32768 ~ +32767
unsigned long	32	4	0 ~ 4294967295
signed long	32	4	−2147483648 ~ +2147483647
float	32	4	±1.176E−38 ~ ±3.40E+38（6 位数字）
double	64	8	±1.176E−38 ~ ±3.40E+38（10 位数字）
指针	24	3	存储空间 0 ~ 65535

（2）整型变量的定义：变量定义的一般形式为

类型说明符　变量名标识符，变量名标识符，...；

例如：

int a，b，c；(a，b，c 为整型变量)

unsigned int x，y；(x，y 为无符号整型变量)

定义变量时，允许在一个类型说明符后，同时定义多个相同类型的变量。这时，各变量名之间用逗号间隔，类型说明符与变量名之间至少用一个空格间隔。

【例 3-4】　整型变量的定义与使用。

```
main ( )
{
  int a, b;
  unsigned c;
  a = 1; b = -4;
  c = a + b;
  printf ( "a = %d, b = %d, c = %d \ n", a, b, c);
}
```

3.2.1.3　实型数据

1. 实型常量的表示方法

实型数据在 C 语言中也称为浮点型。在 C 语言中，它有两种表示形式：分别为十进制小数形式和指数形式。

1）十进制数形式：由数码 0 ~ 9 和小数点组成。

例如：

0.0、24.0、5.189、0.93、90.0、6000.、−227.8670

等均为合法的实数。在这个数中是必须存在小数点的。

2）指数形式：在十进制数的基础上，加阶码标志“e”或“E”和阶码（只能为整数，

可以带正负号）组成。

其一般形式为：

a E n（a 为十进制数，n 为十进制整数）

表示的数值为 $a*10^n$。

例如：

3.1E4（等于 $3.1*10^4$）

1.6E－2（等于 $1.6*10^{-2}$）

0.8E7（等于 $0.8*10^7$）

－1.14E－2（等于 $-1.14*10^{-2}$）

以下不是合法的实数：

385（无小数点）

E6（阶码标志 E 之前无数字）

－1（无阶码标志）

51.－E3（负号位置不对）

2.9E（无阶码）

标准 C 允许浮点数尾部加后缀，“f” 或 “F” 即表示该数为浮点数。如 328f 和 328. 是等价的。

【例 3-5】 通过以下程序例子可以看到其显示输出值都是一样的。

```
main（）{
    printf（“%f\n”，356.）;
    printf（“%f\n”，356）;
    printf（“%f\n”，356f）;
  }
```

2. 实型变量

实型变量主要分为：单精度（float 型）和双精度（double 型），见表 3-2。

表 3-2 float 与 double 类型数据

类型说明符	bit 数（字节数）	有效数字	数的范围
float	32（4）	6～7	$10^{-37}\sim10^{38}$
double	64（8）	15～16	$10^{-307}\sim10^{308}$

在 C 编译器中单精度型占 4 个字节（32 位）内存空间，其数值范围为 3.4E－38～3.4E＋38，只能提供七位有效数字。双精度型占 8 个字节（64 位）内存空间，其数值范围为 1.7E－308～1.7E＋308，提供 16 位有效数字。

关于实型变量定义的方法与整型相同，参考如下：

例如：

float x，y；（x，y 为单精度实型量）

double a，b，c；（a，b，c 为双精度实型量）

3.2.1.4　字符型数据

字符型数据可分为字符常量和字符变量。

1. 字符常量

我们用单引号括起来的一个字符，称为字符常量。

例如：

‘y’、‘f’、‘－’、‘＋’、‘/’

这些都是合法字符常量。

我们在使用字符常量时，需要注意以下几点：

1）字符常量只能用单引号括起来，如果用双引号或其他括号，将出现错误提示。

2）字符常量只能是单个字符，不能出现多个字符。

3）字符可以是字符集中任意字符。但如果将数字定义为字符型常量后，它就不能参加数值运算了。如‘8’和8，仔细看一下是不同的。前者是字符常量，不能参与运算，而后者才是一个整型数据。

2. 转义字符

转义字符在C语言中是一种特殊的字符常量。转义字符是以反斜线“\”开头的字符序列。不同的转义字符具有不同的特定含义，因此我们称它为“转义”字符。例如，我们经常在例题中看到的printf函数中用到的“\n”就是一个转义字符，其意义是“回车换行”。转义字符的作用主要用来表示有些用一般字符不便于表示的语句代码。表3-3列出了C语言常用的转义字符及含义。

表3-3　C语言常用的转义字符及含义

转义字符	转义字符的意义	ASCII代码
\n	回车换行	10
\t	横向跳格（跳到下一输出区）	9
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜线符“\”	92
\’	单引号符	39
\”	双引号符	34
\a	鸣铃	7
\ddd	1～3位八进制数所代表的字符	
\xhh	1～2位十六进制数所代表的字符	

可以说，C语言字符集中的任何一个字符均可用转义字符来表示。表中的\ddd和\xhh正是起到了这个作用。ddd和hh分别为八进制ASCII码和十六进制ASCII码。如\102表示字母“B”，\103表示字母“C”，\XOA表示换行等。

【**例3-6**】　转义字符的使用方法。

```
main ( )
{
   int a, b, c;
   x =5; y =6; z =7;
   printf ( "xy   z\ tde\ rf\ n");
printf ( "hijk\ tL\ bM\ n");
}
```

大家可以想一下，这个程序的输出结果是怎样的。

3. 字符变量

字符变量用来存储单个字符。

字符变量的类型说明符是 char。其定义方法与上面我们所讲的数据类型定义方法一样。例如：

char a，b；

char 数据类型的长度是 1 个字节，通常用于定义处理字符数据的变量或常量。unsigned char 与 signed char 数据类型的区别是有无符号位，如果我们直接使用 char 类型的话，其默认值为 signed char 类型。因为 unsigned char 类型 1 个字节所有的 8 位均来表示数值，而 signed char 类型用字节中的最高位 bit 来表示数据的正负，“0” 表示其为正数，“1” 表示为负数，所以 unsigned char 类型可以表达的数值范围是 0 ~ 255，而 signed char 类型可以表达的数值范围是 −128 ~ +127。

4. 字符串常量

字符串常量是由一对双引号括起的字符序列。例如：“CHINA”，“C program”，“＄ 1.3” 等都是合法的字符串常量。

那么字符串常量与字符常量之间有什么不同之处呢。总结一下，主要有以下几点：

1）字符常量使用单引号括起来，而字符串常量使用双引号括起来。

2）字符常量只能是单个字符，而字符串常量却可以是一个或多个字符。

3）可以把一个字符常量赋给一个字符变量，但反过来把一个字符串常量赋予一个字符变量是不行的。在 C 语言中是没有相应的字符串变量的。

4）字符常量在内存中占 1 个字节的空间。字符串常量在内存中所占字节的大小等于字符串的字节数加 1，这是一个字符串结束的标志位，在 C 语言中，用字符‘\ 0’作为字符串的结束标志，‘\ 0’的 ASCII 码值为 0。

例如：

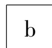
字符串“C program”在内存中所占字节的情况为

C		p	r	o	g	r	a	m	\ 0

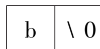
‘\ 0’为字符串“C program”结束的标志位。

字符常量‘b’和字符串常量“b”，从表面上来看，虽然都只有一个字符，但在内存中的存储情况却是不同的。

‘b’在内存中占 1 个字节，其存储情况为：



“b”在内存中占 2 个字节，其存储情况为：



3.2.1.5 变量赋初值

在程序中常常需要对变量赋于一个初值。C 语言可以在做变量定义的同时，给变量赋上初值，我们也称之为变量的初始化操作；也可以在后面的语句中再给变量赋上值。

变量初始化赋初值的一般形式为

类型说明符　变量 1 = 值 1，变量 2 = 值 2，变量 3 = 值 3……；

例如：

int b = 8；

int c，e = 1；

double a = 8.2，b = 6f，c = 0.575；

char ch1 = ‘A’，ch2 = ‘B’；

与其他高级编程语言不同，C 语言在变量定义中不允许出现多个“ = ”赋值符号，如 a = b = c = 3 是非法的。

【例 3-7】

```
main（）
{
  int aa = 3，bb，cc = 5；
  bb = aa + cc；
  printf（“aa = %d，bb = %d，cc = %d\n”，aa，bb，cc）；
}
```

3.2.1.6 各类数值型数据之间的混合运算

变量的数据类型是可以转换的。转换方式有两种，一种是自动类型转换，另一种是强制类型转换。当程序中不同数据类型的变量混合运算时，自动类型转换由编译器自动完成。自动类型转换遵循以下转换规则：

1）若参与运算的数据类型不一样，则先转换成同一数据类型，再进行运算。

2）转换以保证精度不降低的原则进行。

3）所有的浮点运算都是转换成双精度进行的，既使表达式中仅含 float 单精度的数据运算量，也要先转换成 double 型，再作运算处理。

4）在赋值运算中，当赋值号两边的数据类型不同时，赋值号右边的数据类型将转换成左边的数据类型。如果右边的数据类型长度大于左边的数据类型长度时，那它将丢失一部分数据，丢失的部分遵循四舍五入的原则，降低了精度。

类型自动转换的规则如图 3-3 所示。

当 int 和 char 同时进行运算时，则系统先将 char 转为 int 型数据；当 float 与 double 型共同存在时，则先将 float 转为 double 型数据。当 int，unsigned，long，doulbe 类型的数据同时存在时，则其转换高低关系为 int→unsigned→long→double。如果 int 与 long 共存时，则必将 int 转换为 long 类型数据。

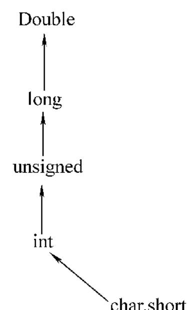


图 3-3　不同数据之间的自动类型转换

【例 3-8】

```
main ( )
  {
float PI =3. 1415;
int s;
int r =6;
s = r * r * PI;
printf ( “面积：s = %d \ n”, s);
   }
```

例 3-8 中，变量 PI 为实型；s，r 为整型。当程序运行到 s = r * r * PI 语句时，变量 r 和变量 PI 都将转换成 double 型数据，其计算结果也为 double 类型，但因为 s 为整型变量，根据自动类型转换原则，其最终结果应该为整型，如出现小数数值，则舍去了小数部分，所以，程序的执行结果，其面积 s = 113。

强制类型转换是通过类型转换运算符来实现的。其一般形式为

（类型名）　（表达式）

其功能是把表达式的运算结果值强制转换成类型名所表示的数据类型。

例如：

(double) a　　将 a 转换为 double 类型数据

(int) (x + y)　将 x + y 的值转换成整型

(float) (5%3) 将 5%3 的值转换成 float 型

在使用强制转换时应注意以下两点：

1）当变量为多个时，类型名和表达式都必须加上括号（单个变量可以不加括号），如：把（int）（x + y）写成（int）x + y，其意义为把 x 转换成 int 型后再与 y 相加；而（int）（x + y）的意义是把 x 和 y 相加，其相加结果再转换成 int 数据类型。

2）强制类型转换时，并没有改变原来变量的类型，而是得到一个所需类型的中间变量，如上例中的变量 x 和变量 y，其本身的数据类型和值并没有发生变化。

【例 3-9】

```
main ( ) {
  float f =1. 39;
  printf ( “ (int) f = %d, f = %f \ n”, (int) f, f);
}
```

从以上程序执行结果可以看出，float 型变量 f 虽然在 printf 函数中强制类型转为 int 型，但它只在运算中起作用，而且是临时的，变量 f 其本身的类型并不改变。因此，（int） f 的值为 1（截去了小数部分数字），而 f 的值仍为 1. 39。

【单片机 C 语言设计小实例 1】 先来写个小程序看看 unsigned char 和 unsigned int 用于延时的不同效果，说明它们的长度是不同的，学习它们的用法。实验中用 LED1 的点亮表明正在用 unsigned int 数值延时，用 LED2 点亮表明正在用 unsigned char 数值延时。电路原理如图 3-4 所示。

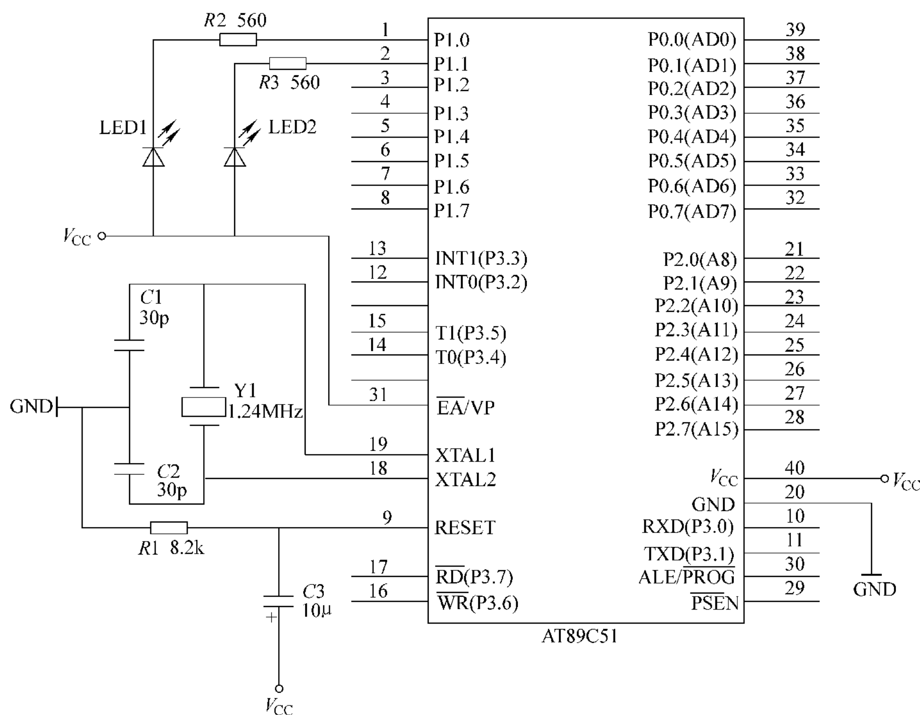


图 3-4 实验电路

实验程序如下：

```
#include <AT89X51. h> //预处理命令
void main (void)    //主函数名
{
unsigned int a;    //定义变量 a 为 unsigned int 类型
unsigned   char   b; //定义变量 b 为 unsigned char 类型
do
{ //do while 组成循环
            for   (a =0;    a <65535;    a + +)
            P1_0   =   0;   //65535 次设 P1. 0 口为低电平，点亮 LED
            P1_0   =   1;   //设 P1. 0 口为高电平，熄灭 LED
            for   (a =0;    a <30000;    a + +);   //空循环
            for   (b =0;    b <255;    b + +)
```

```
            P1_1  =  0;   //255 次设 P1.1 口为低电平，点亮 LED
P1_1  =  1;   //设 P1.1 口为高电平，熄灭 LED
for  (a=0;  a<30000;  a++);   //空循环
        }
while (1);
}
```

执行编译烧写，上电运行你就可以看到结果了。很明显 LED1 点亮的时间要比 LED2 点亮的时间长。请注意，当定义一个变量为特定的数据类型时，在程序使用该变量时，不应使它的值超过数据类型的值域。如本例中的变量 b 不能赋超出 0～255 的值，如 for (b=0; b<255; b++) 改为 for (b=0; b<256; b++)，编译是可以通过的，但运行时就会有问题出现，就是说 b 的值必须是小于 256 的，所以无法跳出循环执行下一句 P1_1 = 1，从而造成死循环。同样，a 的值不应该超出 0～65535 的范围。

【单片机 C 语言设计小实例 2】　跑马灯实验例子，将 P1 口的全部引脚分别驱动 1 个 LED 发光管。电路原理如图 3-5 所示。

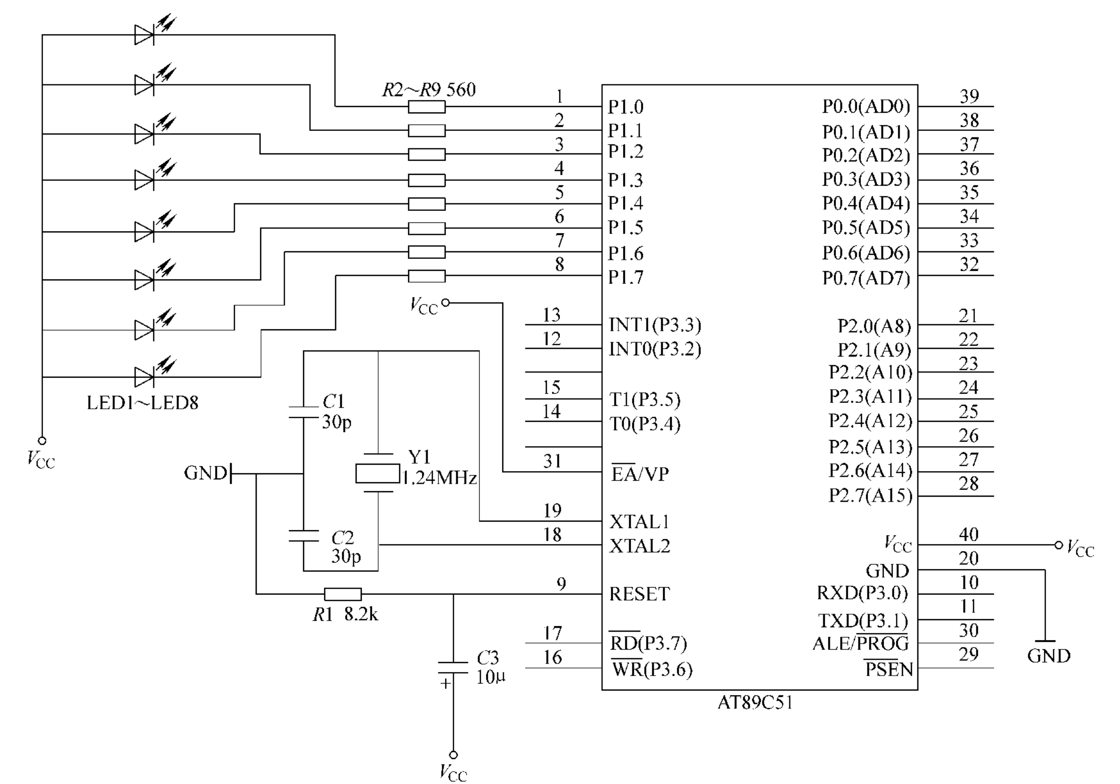


图 3-5　8 路跑马灯电路

程序如下：

```
#include <AT89X51.H> //预处理文件里面定义了特殊寄存器的名称如 P1 口定义为 P1
void main (void)
{
```

```
//定义花样数据
const unsigned char   design [32] = {0xFF, 0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F,
0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE, 0xFF,
0xFF, 0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x0,
0xE7, 0xDB, 0xBD, 0x7E, 0xFF};

unsigned int a; //定义循环用的变量
unsigned char  b; //在 C51 编程中因内存有限尽可能注意变量类型的使用
//尽可能使用少字节的类型, 在大型的程序中很受用
    do {
      for  (b = 0;  b < 32;  b + +)
          {
           for (a = 0;  a < 30000;  a + +);  //延时一段时间
           P1  =  design [b]; //读已定义的花样数据并写花样数据到 P1 口
             }
} while (1);
}
```

程序中的花样数据可以自己去定义, 因这里我们的 LED 要 AT89C51 的引脚 P1 为低电平才会点亮, 所以我们要向 P1 口的各引脚写数据 O 对应连接的 LED 才会被点亮, P1 口的 8 个引脚刚好对应 P1 口特殊寄存器的 8 个二进位。如向 P1 口定数据 0xFE, 转成二进制就是 11111110, 最低位 D0 为 0, 这里引脚 P1.0 输出低电平, LED1 被点亮。如此类推, 大家不难算出自己想要做的效果了。大家编译烧写看看, 效果就会出来, 显示的速度您可以根据需要调整延时 a 的值, 不要超过变量类型的值域就行了。

【单片机 C 语言设计小实例 3】 8 路跑马灯的另一种写法。

程序如下:

```
sfr P1 = 0x90; //这里没有使用预定义文件,
sbit P1_0 = P1 ^ 0; //而是自己定义特殊寄存器
sbit P1_7 = 0x90 ^ 7; //之前我们使用的预定义文件其实就是这个作用
sbit P1_1 = 0x91; //这里分别定义 P1 端口和引脚 P10, P11, P17

void main (void)
{
unsigned  int a;   //定义 a 为 int 变量
unsigned  char  b; //定义 b 为 char 变量
    do {
        for  (a = 0; a < 50000; a + +)
              P1_0  =  0;   //点亮 P1_0
```

```
        for  (a=0; a<50000; a++)
             P1_7  =  0;    //点亮 P1_7
        for  (b=0; b<255; b++)
              {
               for  (a=0; a<10000; a++)
                    P1  =  b;    //用 b 的值来做跑马灯的花样
                 }
            P1  =  255;    //熄灭 P1 上的 LED
            for  (b=0; b<255; b++)
                 {
                   for  (a=0; a<10000; a++)    //P1_1 闪烁
                        P1_1  =  0;
                   for  (a=0; a<10000; a++)
                        P1_1  =  1;
                 }
} while (1);
}
```

3.2.2 算术运算符和算术表达式

C 语言中运算符和表达式数量之多，在高级语言中是少见的。正是丰富的运算符和表达式使 C 语言功能十分完善。这也是 C 语言的主要特点之一。

C 语言的运算符不仅具有不同的优先级，而且还有一个特点，就是它的结合性。在表达式中，各运算量参与运算的先后顺序不仅要遵守运算符优先级别的规定，还要受运算符结合性的制约，以便确定是自左向右进行运算还是自右向左进行运算。这种结合性是其他高级语言的运算符所没有的，因此也增加了 C 语言的复杂性。

3.2.2.1 C 运算符简介

C 语言的运算符可分为以下几类：

1）算术运算符：用于各类数值运算。包括加（+）、减（-）、乘（*）、除（/）、求余（或称模运算,%）、自增（++）、自减（--）共 7 种。

2）关系运算符：用于比较运算。包括大于（>）、小于（<）、等于（==）、大于等于（>=）、小于等于（<=）和不等于（!=）共 6 种。

3）逻辑运算符：用于逻辑运算。包括与（&&）、或（||）、非（!）共 3 种。

4）位操作运算符：参与运算的量，按二进制位进行运算。包括位与（&）、位或（|）、位非（~）、位异或（^）、左移（<<）、右移（>>）共 6 种。

5）赋值运算符：用于赋值运算。分为简单赋值（=）、复合算术赋值（+=，-=，*=，/=，%=）和复合位运算赋值（&=，|=，^=，>>=，<<=）三类共 11 种。

6）条件运算符：这是一个三目运算符，用于条件求值（?:）。

7）逗号运算符：用于把若干表达式组合成一个表达式（,）。

8）指针运算符：用于取内容（*）和取地址（&）两种运算。

9）求字节数运算符：用于计算数据类型所占的字节数（sizeof）。

10）特殊运算符：有括号（），下标［］，成员（→，.）等几种。

3.2.2.2 算术运算符和算术表达式

1. 基本的算术运算符

1）加法运算符“+”：如 a+b

2）减法运算符/负数值符号“-”：如 a-b 或负数 -5

3）乘法运算符“*”：如 a*b

4）除法运算符“/”：如 a/b

5）求余运算符（模运算符）“%”：如 11%3=2，即 11 除以 3 后，余数为 2。

【例 3-10】

```
main ( ) {
  printf ( "%d \ n", 110%3);
}
```

本例输出 110 除以 3 所得的余数 2。

2. 算术表达式和运算符的优先级和结合性

算术表达式——用算术运算符和括号将运算对象连接起来的，符合 C 语言语法的式子，我们称为算术表达式。其运算对象包括常量、变量、函数等。表达式求值运算按运算符的优先级和结合性来进行。

以下是算术表达式的例子：

a+c;

a+b/d*c;

a*（b-c）+（d+e）/f;

a-b/c-9.5+‘d’;

1）运算符的优先级：优先级——当一个运算对象两边都有运算符时，执行运算的先后顺序。优先级高的，则先进行运算。C 语言中，运算符的运算优先级总共分为 15 个等级。1 级最高，15 级最低。在表达式中，优先级较高的比优先级较低的先进行运算。而在一个运算对象两侧的运算符优先级相同时，那就得按运算符的结合性规定进行处理。

2）运算符的结合性：结合性——当一个运算对象两边的运算符出现相等优先级情况下的运算顺序。C 语言中所有运算符的结合性分为两种，左结合性（自左向右）和右结合性（自右向左）。算术运算符的结合性是自左至右，即先左后右。

例如：a+b*c，从这个表达式可以看到，乘法的优先级要高于加减法，因此，先进行 b*c 的运算，然后，再将其积加上 a。

（a-b）*（c+d）+e，该表达式比上面的表达式稍微复杂点，因为括号在算术运算符中的优先级是最高的，故应先做括号内的运算，即完成（a-b）和（c+d）的运算，再

将两个计算结果进行相乘，最后再加上 e。算术运算符的结合性是自左向右的，也称“左结合性”。而自右向左的结合方向称为“右结合性”。最常见的右结合性运算符是赋值运算符。例如 a = b = c，则应先执行 b = c 再执行 a = （b = c） 运算。在编写程序的过程中，希望大家注意其区别，以避免理解错误。

3. 强制类型转换运算符

其一般形式为

（类型名）　（表达式）

其功能是把表达式的运算结果值强制转换成类型名所表示的数据类型。

例如：

（float）a　　把 a 转换为实型数据

（int）（x + y）　把 x + y 的结果转换为整型数据

4. 自增、自减运算符

\+ +：增量运算符。其功能是使变量值自增 1。

\- -：减量运算符。其功能是使变量值自减 1。

这两个运算符是 C 语言中所特有的，使用非常方便，其作用就是对变量做加 1 或减 1 操作。要注意的是变量在符号前或后，其含义是不同的。

i + +　　i 参与运算后，i 的值再自增 1。

i - -　　i 参与运算后，i 的值再自减 1。

粗略地看，+ +i 和 i + + 的作用都相当于 i = i + 1，但 + +i 和 i + + 的不同之处在于 + +i 先执行 i = i + 1，再使用 i 的值；而 i + + 则是先使用 i 的值，再执行 i = i + 1。

例如：

若 i 的值原来为 6，则

k = + +i　k 的值为 7，i 的值也为 7

k = i + +　k 的值为 6，但 i 的值为 7

若 i 的值为 3，表达式 k = （+ +i） + （+ +i） + （+ +i） 的值应该为 18，因为 + +i 最先执行，先对表达式进行扫描，进行 3 次自加（+ +i），则此时 i = 6，之后，表达式应该体现为 k = 6 + 6 + 6 = 18，所以，我们才得出 18 这个数字。若表达式改为 k = （i + +） + （i + +） + （i + +），其最终的 k 值应该是 9，而 i 最终为 6，因此在这个表达式中，先对 i 进行三次相加，再执行三次 i 的自加。

【例 3-11】

```
main ( ) {
int i = 10;
printf ( "%d \ n", + +i);
printf ( "%d \ n", - -i);
printf ( "%d \ n", i + +);
printf ( "%d \ n", i - -);
printf ( "%d \ n", -i + +);
```

```
  printf ( "%d\n", -i--);
}
```

i 的初值为 10，第 2 行 i 加 1 后输出为 11；第 3 行减 1 后输出为 10；第 4 行输出 i 为 10 之后再本身加 1（此时 i 为 11）；第 5 行输出 i 为 11 之后再本身减 1（此时 i 为 10）；第 6 行输出 -10 之后再加 1（此时 i 为 11），第 7 行输出 -11 之后再减 1（此时 i 为 10）。

3.2.2.3　赋值运算符和赋值表达式

1. 赋值运算符

赋值运算符"="的作用就是将一个数据赋给一个变量。

其一般形式为：

变量 = 表达式

例如：

c = a + b

w = sin（x） + cos（y）

赋值表达式的作用是将表达式的计算结果值赋给"="左边的变量。由于赋值运算符具有右结合性。因此

如果有：x = y = z = 28

可理解为

x = （y = （z = 28））

2. 复合的赋值运算符

我们经常会看到一些源程序代码中，出现以下的运算符，这样做的好处是简化了程序，提高了 C 程序代码的编译效率。

+=，-=，*=，/=，%=，<<=，>>=，&=，^=，|=

其效果如下：

a+=b　相当于 a=a+b

a-=b　相当于 a=a-b

a*=b　相当于 a=a*b

a/=b　相当于 a=a/b

a%=b　相当于 a=a%b

a<<=b　相当于 a=a<<b

a>>=b　相当于 a=a>>b

复合赋值符这种写法，对初学者来说可能不太习惯，但非常有利于编译处理，能提高编译效率并产生高质量的目标代码。

3.2.2.4　逗号运算符和逗号表达式

在 C 语言中，提供了一种特殊的运算符，它就是逗号运算符——","。用逗号运算符将两个表达式连接起来组成一个表达式，称为逗号表达式。

其一般形式为

表达式 1，表达式 2，表达式 3，……表达式 *n*

逗号表达式的求解过程是，从左到右计算出各个表达式的值，而整个逗号运算表达式的值等于最右边那一个表达式的值，就是“表达式 *n*”的值。在大部分情况下，使用逗号表达式的目的只是为了分别得到各个表达式的值，而并不一定要得到使用整个逗号表达式的值。还需要注意的是，并不是程序中任何位置上出现的逗号都认为是逗号运算符。如函数中的参数，同数据类型变量的定义中的逗号只是用来起到间隔作用，而并不是逗号运算符。

【例 3-12】

```
main ( ) {
int a =1, b =2, c =3, x, y;
y = (x =a +b), (b +c);
printf ( "y =%d, x =%d", y, x);
}
```

例 3-12 中，y 等于整个逗号表达式的值，也就是（b + c）的值，x 是第一个表达式的值。对于逗号表达式的使用过程中还请注意以下两点：

1）逗号表达式可以进行嵌套使用。

例如：

表达式 1，（表达式 2，表达式 3）

可以把（表达式 2，表达式 3）看成是一个逗号表达式。

2）程序中使用逗号表达式，通常是要分别求逗号表达式内各表达式的值，但并不一定求得整个逗号表达式的值。

3.2.3 关系运算符和表达式

在程序中我们经常会需要比较两个变量的大小关系，以便做出程序的不同功能选择，我们将比较两个数据量的运算符称为关系运算符。

3.2.3.1 关系运算符及其优先次序

在 C 语言中有以下关系运算符：

1）< 小于

2）< = 小于或等于

3）> 大于

4）> = 大于或等于

5）= = 等于

6）! = 不等于

对于关系运算符，我们同样也并不陌生，C 语言中有 6 种关系运算符，这些运算符的意义看上去也非常直观。或许你从来没用 C 语言写过程序，那么对前面 4 个关系运算符一定是再熟悉不过的了。而“= =”在 VB 或 PASCAL 等语言中是用“=”、“! =”则是用

“not”。关系运算符都是双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，高于赋值运算符。在6种关系运算符中，<，<=，>，>=的优先级相同，高于==和!=，==和!=的优先级相同。

3.2.3.2 关系表达式

当两个表达式用关系运算符连接起来时，我们称之为关系表达式。关系表达式通常是用来判别某个条件是否满足。要注意的是用关系运算符的运算结果只有“0”和“1”两种，也就是逻辑的真与假，当指定的条件满足时结果为“1”，不满足时结果为“0”。

关系表达式的一般形式为

表达式 关系运算符 表达式

例如：

a+b>c

a>9

都是合法的关系表达式，由于表达式也可以是关系表达式，因此表达式也允许出现嵌套的情况。例如：

a<（b<c）

x!=（y==z）

关系表达式为“真”时，用“1”表示；关系表达式为“假”时，用“0”表示。

如：

2>1 的值为“真”，即为“1”。

（a=3）>（b=5）由于3>5不成立，故其值为假，即为0。

【例 3-13】

```
main () {
    char c = 'k';
    int i=2, j=4, k=6;
    float x=2e+6, y=0.65;
    printf ("%d,%d\n", 'a' +5<c, -i-2*j>=k+1);
    printf ("%d,%d\n", 1<j<5, x-5.25<=x+y);
    printf ("%d,%d\n", i+j+k==-2*j, k==j==i+5);
}
```

在本例中打印出了各种关系运算符的值。以上部分printf语句中，字符变量是以它相应的ASCII码值参与运算的。对于含多个关系运算符的表达式，如语句中出现“k==j==i+5”的情况，则C编译器会根据运算符的左结合性，先执行k==j，该式不成立，其值为“0”；再执行0==i+5，因为i=2，所以等式也不成立，故表达式值最终的值为“0”。

【单片机C语言设计小实例4】 输入两个数，判断两数之间的大小关系。

```
#include   <AT89X51. H>
#include  <stdio. h>

void main (void)
{
int x, y;
SCON   =   0x50; //串口方式 1, 允许接收 TMOD  = 0x20; //定时器 1 定时方式 2
TH1  = 0xE8;    //11. 0592MHz 1200 波特率 TL1  = 0xE8;
TI   =   1;
TR1  = 1; //启动定时器

while (1)
    {      printf ( "请输入两个整型数字, X 和 Y \ n"); //显示
           scanf ( "% d% d", &x, &y);          //输入
           if   (x    <   y)
                printf ( "X < Y \ n");    //当 X 小于 Y 时
           else      //当 X 不小于 Y 时再作判断
                {
                    if   (x   = =   y)
                         printf ( "X = Y \ n");    //当 X 等于 Y 时
           else
                         printf ( "X > Y \ n");    //当 X 大于 Y 时
           }
     }
}
```

3. 2. 4　逻辑运算符和表达式

3. 2. 4. 1　逻辑运算符及其优先次序

关系运算符所能反映的是两个表达式之间的大小关系，而逻辑运算符则是用于求条件式的逻辑值，用逻辑运算符将关系表达式或逻辑量连接起来的就是逻辑表达式。也许你会对为什么“逻辑运算符将关系表达式连接起来就是逻辑表达式了”这一描述有疑惑的地方。其实在前面部分我们已经说过，“用关系运算符的运算结果只有“0”和“1”两种，也就是逻辑的真和假”，换句话说也就是逻辑量，而逻辑运算符就用于对逻辑量运算的表达。

C 语言中提供了 3 种逻辑运算符：

1）&&　“与”运算

2）| |　“或”运算

3）!　“非”运算

与运算符“&&”和或运算符“| |”均为双目运算符，具有左结合特性。非运算符

“!”为单目运算符，具有右结合特性。

逻辑运算符和其他运算符优先级的关系如图3-6所示。

“&&”和“||”低于关系运算符，“!”高于算术运算符。

逻辑运算符也有优先级别，从高到低排列，依次如下：!（逻辑非）→&&（逻辑与）→||（逻辑或）。逻辑非的优先级最高，逻辑或的优先级最低。

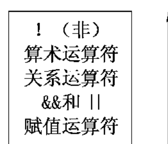


图3-6　运算符优先级

按照运算符的优先顺序可以得出

a > b && x < y　　等价于　　（a > b）&&（x < y）

! b = = c || d < e　　等价于　　（（! b）= = c）||（d < e）

a + b > c&&x + y < z　　等价于　　（（a + b）> c）&&（（x + y）< z）

再来看一个例子，如有　! True　||　False && True

按逻辑运算的优先级别来分析则得到（True 代表真，False 代表假）

```
! True   ||   False   &&   True
False    ||   False   &&   True   //! Ture 先运算得 False
False    ||   False               //False  &&  True 运算得 False
False                             //最终 False  ||  False 得 False
```

3.2.4.2　逻辑运算的值

逻辑运算的结果值分为“真”和“假”两种，用“1”和“0”来表示。其求值规则如下：

1）与运算 &&：参与运算的两个量都为真时，结果才为真，否则为假。

例如：

4 >0 && 9 >2

因为4 >0 为真，9 >2 也为真，两边同时满足真，所以它们相“与”的结果也为真。

2）或运算||：参与运算的两个量只要有一个为真时，结果就为真。两个量都为假时，结果为假。

例如：

4 >0||5 >8

虽然5 >8 为假，但因为4 >0 为真，所以其最终结果也就为真。

3）非运算!：参与运算量为真时，结果为假；参与运算量为假时，结果为真。

例如：

!（5 >0）

其结果为假。

虽然C语言在进行逻辑运算时，以“1”代表“真”，“0”代表“假”，但是否意味着“真”就是“1”，“假”就是“0”呢？其实在C语言中并不是这样的，C语言规定，“0”代表“假”，而以非“0”值代表“真”，这点请大家注意区别。

例如：

由于 1 和 9 均为非“0”因此 1&&9 的值为“真”，即为“1”。

又如：

9| |0 的值为“真”，即为“1”。

3.2.4.3 逻辑表达式

逻辑表达式的一般形式为

表达式1　逻辑运算符　表达式2

与关系表达式类似，逻辑表达式同样也可以出现嵌套的情况。

例如：

(x&&y) &&z

根据逻辑运算符的左结合性，上面式子等价于

a&&b&&c

逻辑表达式的值就是式子中所有逻辑运算的最后结果值，用“1”和“0”分别代表“真”和“假。

逻辑“与”，就是当条件式 1“与”条件式 2 都为真时，结果为真（非 0 值），否则为假（0 值）。也就是说编译器会先对条件式 1 进行判断，如果为真（非 0 值），则继续对条件式 2 进行判断，当结果为真时，逻辑运算的结果为真（值为 1），如果结果为假时，逻辑运算的结果为假（0 值）。如果条件式 1 的逻辑值为假时，那就不用再判断条件式 2 了，而直接给出运算结果为假，即值为“0”。

逻辑“或”，是指只要两个运算条件中有一个为“真”时，运算结果就为“真”，只有当条件式都为假时，逻辑运算结果才为假。

逻辑“非”，则是把逻辑运算结果值取反，也就是说如果两个条件式的运算值为真，进行逻辑“非”运算后则结果变为假；条件式运算值为假时，那么最后逻辑结果为真。

【例 3-14】

```
main ( ) {
    char c = 'k';
    int i =1, j =3, k =5;
    float x =3e +6, y =0.25;
    printf ("%d,%d\n",! x*! y,!!! x);
    printf ("%d,%d\n", x| | i&&j -3, i <j&&x <y);
    printf ("%d,%d\n", i = =5&&c&& (j =8), x +y| | i +j +k);
}
```

例 3-14 中！x 和！y 的值分别为 0，所以！x＊！y 也为 0，故其输出值为 0。由于 x 为非 0，故!!! x 对 0 做了三次的逻辑非操作，最终的逻辑值仍为 0。对于式子 x| | i && j -3，先计算 j -3 的值为非 0，再求 i && j -3 的逻辑值为 1，因此 x| | i&&j -3 的逻辑值为 1。

对于式子 i < j&&x < y，由于 i < j 的值为 1，而 x < y 为 0，故表达式的值为“1”和“0”相与，最终等于 0。对于式子 i = = 5&&c&&（j = 8），由于 i = = 5 为假，即值为 0，该表达式由两个逻辑与运算组成，而当第一个表达式的值为“0”时，就不会再去判断后面的表达式的值了，所以整个表达式的值为 0。对于式子 x + y | | i + j + k 由于 x + y 的值为非 0，故整个表达式的值为 1。

【单片机 C 语言设计小实例 5】 如有 ！True | | False && True

按逻辑运算的优先级别来分析则得到（True 代表真，False 代表假）

```
！True  | |  False  &&  True
False  | |  False  &&  True  //！Ture 先运算得 False
False  | |  False            //False  &&  True 运算得 False
False                        //最终 False  | |  False 得 False
```

下面我们用程序语言来表达：

```
#include  <AT89X51.H>

#include <stdio.h>
void main（void）
{
unsigned  char  True = 1; //定义
unsigned  char  False = 0;

SCON  =  0x50; //串口方式 1，允许接收 TMOD = 0x20; //定时器 1 定时方式 2
TH1 = 0xE8;   //11.0592MHz 1200 波特率 TL1 = 0xE8;
TI  =  1;
TR1 = 1; //启动定时器

if  （! True | | False && True）
            printf（“True\n”）;   //当结果为真时
else
            printf（“False\n”）;   //结果为假时
}
```

第 4 章　分支与循环控制

4.1　if 语句

C 语言是一种结构化程序设计语言。它不允许交叉程序流程的存在，而是以模块为基本单位。一个程序只有一个入口和一个出口，不允许有突然的中途插入或退出。

结构化程序由若干模块组成，每个模块中包含着若干个基本结构，而每个基本结构中可以有若干条语句。

归纳起来，C 语言有 3 种基本结构：

①　顺序结构；

②　选择结构；

③　循环结构。

1. 顺序结构及其流程图

顺序结构是一种最基本、最简单的编程结构。在这种结构中，程序由低地址向高地址顺序执行指令代码。如图 4-1 所示，程序先执行 A 操作，再执行 B 操作，两者是顺序执行的关系。

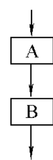


图 4-1　顺序结构流程图

2. 选择结构及其流程图

如果计算机只能做像顺序结构那样简单的基本操作，则它的用途十分有限。计算机功能强大的原因就在于它具有决策能力或者说具有选择能力。如图 4-2 所示。

- 依靠条件选择开关，打开或关闭水泵。
- 如果上面的这种操作重复执行了 22 次，那么继续执行下面另一个操作。
- 连续监测一个信号。这个信号指示语言芯片可以接受下一个字的代码。

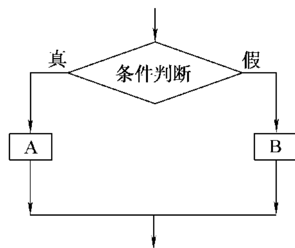


图 4-2　选择结构流程图

C 语言的条件语句与其他语言一样，“如果 XX 就 XX”或是“如果 XX 就 XX 否则 XX”。也就是当条件符合时就执行语句。条件语句又被称为分支语句，它根据给定的条件进行判断，以决定执行某个分支程序段。也有人会称为判断语句，其关键字是由 if 构成，这在众多的高级语言中都是基本相同的。C 语言提供了 3 种形式的条件语句。

4.1.1　if 语句的 3 种形式

1. 第 1 种形式为基本形式：if

```
if（表达式）语句
```

其语义是：如果表达式的值为真，则执行其后的语句，否则就不执行该语句。其过程如

图 4-3 所示。

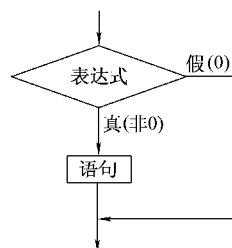


图 4-3　第 1 种 if 语句执行情况

【例 4-1】

```
main ( )
{
    int a, b, min;
    printf ( "\ n 请输入两个数:");
    scanf ( "%d%d", &a, &b);
    min = a;
    if (min > b) min = b;
    printf ( "最小的一个数为 = %d", min);
}
```

例 4-1 中，输入两个数 a，b。把 a 先赋予变量 min，再用 if 语句判别 min 和 b 的大小，如 min 大于 b，则把 b 赋予 min。所以 min 中放的总是小的数，最后将其值进行输出。

2. 第 2 种形式为：if-else

```
if (表达式)
    语句 1;
else
    语句 2;
```

其语义是：如果表达式的值为真，则执行语句 1，否则执行语句 2 。

其执行过程如图 4-4 所示。

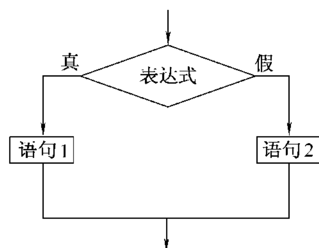


图 4-4　第 2 种 if 语句执行情况

【例 4-2】

```
main ( )
{
int a, b, c;
printf ( "请输入两个数");
scanf ( "%d%d", &a, &b);
if (a = = b)
  {
        c = 1;
    else
        c = 2;
      }
printf ( "c = %d", c);
```

}

例 4-2 中，输入两个整数，用 if-else 语句判别，当 a 等于 b 时，c = 1；否则，c = 2。通过 printf 语句输出变量 c 的值。

3. 第 3 种形式为：if-else-if

前面两种形式的 if 语句一般都用于两个分支的情况。当有多个分支选择时，可采用第 3 种形式，即 if-else-if 语句，其一般形式为

```
if（表达式 1）
    语句 1;
elseif（表达式 2）
    语句 2;
else if（表达式 3）
    语句 3;
  …
else if（表达式 m）
    语句 m;
else
    语句 n;
```

其语义是：先判断表达式 1 的值，如果为真，则执行语句 1；如果表达式 1 的值为假，则再判断表达式 2 的值，如果表达式 2 的值为真，则执行语句 2；否则继续判断表达式 3 的值，就这样依次判断表达式的值，当出现某个值为真时，则执行其后面对应的语句，语句执行完后跳到整个 if 语句之外继续执行程序代码。如果所有的表达式都为假，那么执行语句 n，即最后一个 else 后面的语句，然后再继续执行后面的程序代码。if-else-if 语句的执行过程如图 4-5 所示。

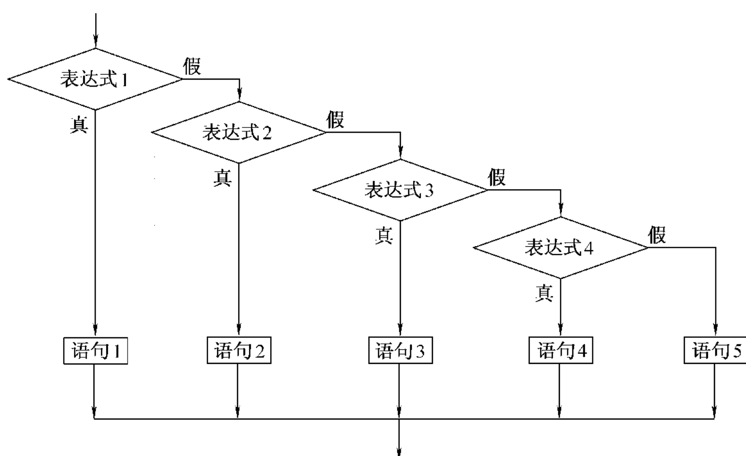


图 4-5　第 3 种 if 语句执行情况

【例 4-3】

```
#include "stdio. h"
main ( ) {
    char c;
    printf ( "输入一个字符:");
    c = getchar ( );
    if (c < 32)
    printf ( "这是一个控制字符\ n");
    else if (c > = '0' &&c < = '9')
    printf ( "这是一个数字\ n");
    else if (c > = 'A' &&c < = 'Z')
    printf ( "这是一个大写字母\ n");
    else if (c > = 'a' &&c < = 'z')
    printf ( "这是一个小写字母\ n");
    else
    printf ( "这是其它类型的字符\ n");
}
```

例 4-3 中，程序的功能是判断键盘输入字符的类别。其原理是根据输入字符的 ASCII 码值来进行判别字符类型。我们查一下 ASCII 码表便可知 ASCII 码值小于 32 的为控制字符。在“0”和“9”之间的为数字，在“A”和“Z”之间为大写字母，在“a”和“z”之间为小写字母，其余则为其他字符。这是一个多分支选择问题的典型应用，用 if-else-if 语句来实现，判断输入的字符 ASCII 码所在的范围，分别给出不同的输出提示信息。例如输入为“x”，输出显示“这是一个小写字母”。

4. 在使用 if 语句中应注意的问题

1）在 3 种形式的 if 语句中，在 if 关键字之后均为表达式，它除了是常见的关系表达式或逻辑表达式外，也允许是其他类型的数据，如整型、实型、字符等。

例如：

```
if (x = 8) 语句;
if (x) 语句;
```

如以上式子中（x = 8）和（x）都是允许的，只要表达式的值为非 0 值，即为“真”，那就执行 if 后面的语句。

如将上面的式子改为：

```
if (x = y)
   printf ( "%d", x);
else
   printf ( "x 的值为 0");
```

该程序段的意思是，把 y 的值赋给 x，如果 x 不为 0，即表达式的值不为 0，那么就执行 printf（“% d”，x）；语句，打印输出 x 的值，否则输出“x 的值为 0”的字符串。

2）与 BASIC 语言不同的是，在 if 语句中，条件判断语句必须用括号将表达式括起来，而且在语句后面必须加分号。

3）在所有的 if 语句形式中，if 后面跟着的语句应为单条语句。如果想在满足条件分支时执行多条语句，那么，我们必须把这若干条语句用“｛｝”括起来，我们称此为复合语句。C 语言中，每一条语句末尾需要加上“;”，在复合语句中，需要注意的是，分号应加在“｝”之内，而不能加在“｝”外面。

例如：

```
if（x>y）
  {a++;
 printf（“x>y”）;}
else
  {a--;
 printf（“x<=y”）;}
```

该程序段意义：如果 x>y，变量 a 自加 1，打印输出“x>y”；如果 x<0 或 x=0，变量 a 自减 1，打印输出“x<=y”。

在 C 语言中有不少的括号，如｛｝，[ ]，（）等，确实会让一些初入门的朋友不解。在 VB 等一些语言中同一个（）号会有不同的作用，它可以用于组合若干条语句形成功能块，可以用做数组的下标等。而在 C 语言中括号的分工较为明显，｛｝号是用于将若干条语句组合在一起形成一种功能块，这种由若干条语句组合而成的语句就叫复合语句。复合语句之间用｛｝分隔，而它内部的各条语句还是需要以分号“;”结束。复合语句是允许嵌套的，也是就是在｛｝中的｛｝也是复合语句。复合语句在程序运行时，｛｝中的各行单语句是依顺序执行的。C 语言中可以将复合语句视为一条单语句，也就是说在语法上等同于一条单语句。对于一个函数而言，函数体就是一个复合语句，因此复合语句中不单可以用可执行语句组成，还可以用变量定义语句组成。要注意的是在复合语句中所定义的变量，称为局部变量，所谓局部变量就是指它的有效范围只在复合语句中，而函数也算是复合语句，所以函数内定义的变量有效范围也只在函数内部。

4.1.2 if 语句的嵌套

如果 if 语句后面的执行语句中又出现了 if 语句时，这就是 if 语句的嵌套使用。

其一般形式可表示如下

```
if（表达式）
if 语句;
```

或者为

```
if（表达式）
  if 语句;
else
  if 语句;
```

嵌套部分内容可能是 if 语句，也有可能是 if-else 类型的语句，当然也可能出现很多个 if

和 else 重叠的情况，在这时，我们需要特别注意 if 和 else 的配对使用问题，自己的脑中要清晰地知道程序代码中后面的 else 语句是相应前面的哪个 if 语句。

下面我们来看一个例子：

```
if（表达式 1）
if（表达式 2）
   语句 1；
else
   语句 2；
```

大家看了以上的 if 程序结构后，想一想，其中的 else 究竟是与哪一个 if 配对呢？是与表达式 1 配对，还是与表达式 2 配对呢？

或许你有可能将其理解成这样：

```
if（表达式 1）
   if（表达式 2）
      语句 1；
   else
      语句 2；
```

也许你也有可能理解成这样：

```
if（表达式 1）
   if（表达式 2）
      语句 1；
   else
      语句 2；
```

那么究竟哪种理解才是对的呢？为了避免这种二义性情况，C 语言规定，程序中的 else 语句总是与它前面最近的 if 相配对，因此对上述例子前一种情况理解是正确的。

【例 4-4】 比较两个数的大小关系。

```
main（）{
   int x，y；
   printf（“请输入两个数字：”）；
   scanf（“%d%d”，&x，&y）；
   if（x！=y）
   if（x>y）    printf（“第一个数字比第二个数字大\n”）；
   else         printf（“第一个数字比第二个数字小\n”）；
   else         printf（“第一个数字等于第二个数字\n”）；
}
```

例4-4 中用了 if 语句的嵌套结构。采用嵌套结构的目的是为了进行多分支选择，即两个数字比较大小，将会有3 种情况出现，x > y、x < y 或 x = y。当 x 不等于 y 的条件满足时，再去执行（x > y）的 if 条件表达式。

【例 4-5】 计算函数

```
        1      x > 0
        y = 0  x = 0
        -1     x < 0
main（ ）
{
    float x，y；
    printf（ “请输入两个数：”）；
    scanf（ “%f”，&x）；
    if（x > =0）
        if（x >0）
            y  =1；
        else
            y  =0；
    else
            y  = -1；
    printf（ “y  =  f  \  n”，y ）；
}
```

这是由 if else 语句组成的嵌套，用来实现多个条件分支，使用时应注意 if 和 else 的配对使用，要是缺了一半编译器就会报语法出错，else 语句总是与最近的 if 语句相配对。一般条件语句用于单一条件或少量的分支数，如果是数量多的分支时，则更多的会用到开关语句——SWITCH 语句。如果使用条件语句来编写超过 3 个以上的分支程序的话，会使程序变得不太清晰易读。

4.2 条件运算符和条件表达式

如果在条件语句中，只执行单个的赋值语句时，常可使用条件表达式来实现。不但使程序简洁，也提高了运行效率。

C 语言中有一个三目运算符，它就是“?:”条件运算符，它要求有 3 个运算对象。它可以把 3 个表达式连接构成 1 个条件表达式。

由条件运算符组成条件表达式的一般形式为

表达式 1？ 表达式 2： 表达式 3

其求值规则为：当逻辑表达式的值为真时（非 0 值）时，整个表达式的值为表达式 2

的值；当逻辑表达式的值为假（值为0）时，整个表达式的值为表达式3的值。要注意的是条件表达式中逻辑表达式的类型可以与表达式2和表达式3的类型不一样。

条件表达式通常用于赋值语句之中。

例如条件语句：

```
if (x < y) min = x;
else min = y;
```

可用条件表达式简写为

```
min = (x<y)? x: y;
```

执行该语句的语义是：如x<y为真，则把x赋予变量min，否则就把y赋给min。

很明显这样写的结果和含义都和上面的一段程序是一样的，但是代码却比上一段程序要少很多，编译的效率相对来说也就高些，但有着和复合赋值表达式一样的缺点，就是可读性相对较差。在实际应用时要根据自己习惯来使用，这样可以有助于程序的调试和编写，也便于程序日后的修改与更新。

使用条件表达式时，还应注意以下几点：

1）条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值符。

因此

```
min = (x>y)? x: y
```

可以去掉括号而等价为

```
min = x>y? x: y
```

2）条件运算符“?”和“:”是一个整体，即一个运算符，我们不能将其分开单独使用。

3）条件运算符的结合方向是自右向左的。

例如：

```
a<b? a: c<d? c: d
```

等价为

```
a<b? a: (c<d? c: d)
```

它类似于if语句的嵌套使用，这也就是条件表达式嵌套的情形，即其中的表达式3又是1个条件表达式。

【例4-6】 输入一个字符。判别它是否大写字母，如果是，将其转换为小写，否则不转换。然后输出最后得到的字符。

```
main ()
{ char ch;
  scanf ("%c", &ch);
  ch = (ch >= 'A' && ch <= 'Z') ? (ch+32): ch;
  printf ("%c", ch);
}
```

4.3 switch 语句

学习了条件语句，用多个条件语句可以实现多方向条件分支，但是发现使用过多的条件语句实现多方向分支会使条件语句嵌套过多，程序冗长，这样读起来也很不好读。这时使用开关语句同样可以达到处理多分支选择的目的，又可以使程序结构清晰。其一般形式为

```
switch（表达式）{
case 常量表达式 1：语句 1；break；
case 常量表达式 2：语句 2；break；
…
case 常量表达式 n：语句 n；break；
default：          语句 n+1；
}
```

其语义是：计算 switch 后面表达式的值，并将其作为条件与 case 后面的各个常量表达式的值相比，如果相等时则执行 case 后面的语句，再执行 break（间断语句）语句，跳出 switch 语句结构；如果 case 后面没有和条件相等的值时就执行 default 后的语句。如果当没有符合的条件时，不做任何处理，那可以不写 default 语句，default 语句只是程序不满足所有 case 语句条件情况下的一个默认情况执行语句。

【例 4-7】

```
main（）{
    int a；
    printf（“请输入一个十进制整数：”）；
    scanf（“%d”，&a）；
    switch（a）{
    case 1：printf（“一\n”）；
    case 2：printf（“二\n”）；
    case 3：printf（“三\n”）；
    case 4：printf（“四\n”）；
    case 5：printf（“五\n”）；
    case 6：printf（“六\n”）；
    case 7：printf（“七\n”）；
    default：printf（“其他数字\n”）；
    }
}
```

本程序是要求输入一个数字，输出其中文书写数字。但是当输入 2 之后，却执行了 case2 以及以后的所有语句，输出了“二”及以后的所有单词，这当然并非我们的本意所在。为什么会出现这种情况呢？这恰恰反应了 switch 语句的一个特点。在 switch 语句中，“case 常量表达式”只相当于一个语句标号，表达式的值和某标号相等则转向该标号执行，但不能在执行完该标号的语句后自动跳出整个 switch 语句，所以出现了继续执行所有后面 case 语句的情况。这是与前面介绍的 if 语句完全不同的，应特别注意。为了避免上述情况，C 语言还提供了一种 break 语句，专门用于跳出 switch 语句，break 语句只有关键字 break，没有参数。在后面还将详细介绍。修改例题的程序，在每一 case 语句之后增加 break 语句，使每一次执行之后均可跳出 switch 语句，从而避免输出不应有的结果。

【例 4-8】 输入月份，打印 1999 年该月有几天。网上转载

程序如下：

```
#include <stdio.h>
main ( )
{
int month;
int day;
printf ( “please input the month number :”);
scanf ( “%d”, &month);
switch (month)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: day =31;
        break;
    case 4:
    case 6:
    case 9:
    case 11: day =30;
        break;
    case 2: day =28;
        break;
    default : day = -1;
}
if day = -1
```

```
        printf ( "Invalid month input ! \ n");
    else
        printf ( "1999. %d has %d days \ n", month, day);
}
```

在使用 switch 语句时还应注意以下几点：

1）在 case 后的各常量表达式的值都应该是不一样的，否则会出现错误。

2）在 case 后，允许出现多条语句，可以不用 {} 括起来。

3）各 case 和 default 语句位置的先后顺序可以改变，而不会影响程序执行结果。

4）default 子句可以省略不写。

【例 4-9】 输入 3 个整数，输出最大数和最小数。

```
main ( ) {
    int a, b, c, max, min;
    printf ( "input three numbers:");
    scanf ( "%d%d%d", &a, &b, &c);
    if (a > b)
       {max = a; min = b;}
    else
       {max = b; min = a;}
    if (max < c)
      max = c;
    else
      if (min > c)
    min = c;
      printf ( "max = %d\ nmin = %d", max, min);
}
```

本程序中，首先比较输入的 a，b 的大小，并把大数装入 max，小数装入 min 中，然后再与 c 比较，若 max < c，则把 c 赋予 max；如果 c < min，则把 c 赋予 min。因此 max 内总是最大数，而 min 内总是最小数。最后输出 max 和 min 的值即可。

【例 4-10】 计算器程序。用户输入运算数和四则运算符，输出计算结果。

```
main ( ) {
    float a, b;
    char c;
    printf ( "input expression: a + ( -, *, /) b \ n");
    scanf ( "%f%c%f", &a, &c, &b);
    switch (c) {
```

```
        case '+': printf ("%f\n", a+b); break;
        case '-': printf ("%f\n", a-b); break;
        case '*': printf ("%f\n", a*b); break;
        case '/': printf ("%f\n", a/b); break;
        default: printf ("input error\n");
    }
}
```

本例可用于四则运算求值。switch 语句用于判断运算符，然后输出运算值。当输入运算符不是+，-，*，/时给出错误提示。

4.4 循环控制

4.4.1 概述

几乎每个程序都会用到循环语句，它的作用就是用来实现语句多次的反复执行操作。如一个 频率为 12MHz 的 51 单片机应用电路中，要求实现 1ms 的延时，那它就要执行 1000 次空语句才可以达到延时的目的（当然可以使用定时器来做，我们将在后面介绍定时器的使用），如果手工写 1000 条空语句那是非常麻烦的事情，难以想象，其次就是要占用很多的存储空间。我们知道这 1000 条空语句，都是一样的语句——即一条空语句重复执行 1000 次，因此我们就可以用循环语句去写，这样不但使程序结构简洁，查看代码显示直观，而且使编译的效率大大提高。在 C 语言中构成循环控制的语句有 goto，while，do-while，for 语句。这些语句都能起到循环作用，但其具体的用法会有一些区别，我们根据具体的情况来选用哪种循环语句。

C 语言提供了多种循环语句，可以组成各种不同形式的循环结构。

1）goto 语句和 if 语句构成循环；

2）while 语句；

3）do-while 语句；

4）for 语句。

4.4.2 goto 语句以及用 goto 语句构成循环

goto 语句在很多高级语言中都会有，和 BASIC 语言中的一样，这个并不陌生。它是一个无条件的转向语句，只要执行到这个语句，程序指针就会跳转到 goto 后的标号所在的程序段。

goto 语句的使用格式为

```
goto 语句标号;
```

其中标号是一个有效的标识符，这个标识符与一个“:”相连接，一起出现在函数内某处，执行 goto 语句后，程序将跳转到该标号处并执行其后的语句。另外标号必须与 goto 语句

同处于一个函数中，但可以不在一个循环层中。通常 goto 语句与 if 条件语句连用，当满足某一条件时，程序跳到标号处运行。

结构化程序设计应尽量不用 goto 语句，主要因为它将使程序层次不清，不易读懂，但在程序多层嵌套退出时，使用 goto 语句则比较合理、方便。

【例 4-11】

```
void main (void)
{
unsigned char a;
loop: a++;
if (a==100) goto end;
goto loop;
end: ;
}
```

例 4-11 中说明了 goto 语句的用法，实际编写很少使用这样的手法，在此我们仅用它来说明语句的用法。这段程序的意思是在程序开始处用标识符“loop:”标识，表示程序的开始，“end:”标识程序的结束，标识符的定义应遵循 C 语言标识符定义原则，不能用 C 语言系统中的关键字，也不能和其他变量或函数名相同，否则编译时将出现错误提示。程序执行 a++，a 的值加 1，当 a 等于 100 时，程序就会跳到 end 标识处，即执行一条空语句后，结束程序运行，否则跳回到 loop 标识处继续执行 a++ 语句，直到 a 等于 100 才结束程序运行。例 4-11 说明 goto 不但可以无条件的转向，也可以和 if 语句构成一个循环结构，不过这样的用法并不太常见，goto 语句用的最多的是用它来跳出多重循环，不过它只可以从内层循环跳到外层循环，不能从外层循环跳到内层循环。过多的使用 goto 语句会使程序结构不清晰，过多的跳转就使程序变得又像汇编语言的风格，而失去了 C 程序模块化的优点。

4.4.3 while 语句

while 语句的意思很容易理解，其英语单词解释的意思是“当…的时候”，在这里我们可以理解为“当条件为真的时候就执行后面的语句”。

while 语句的一般形式为

while（表达式）语句

其中表达式是循环条件，语句为循环体。

while 语句的语义是：计算表达式的值，当值为真（非 0）时，执行循环体语句。

使用 while 语句时要注意当条件表达式为真时，它执行后面的语句，执行一次完成之后再次回到 while 执行条件判断，如果为真，则重复执行语句，为假时退出循环体。当条件一开始就为假时，那么 while 后面的循环体一次都不会被执行就退出整个循环。

While 语句的执行过程如图 4-6 所示。

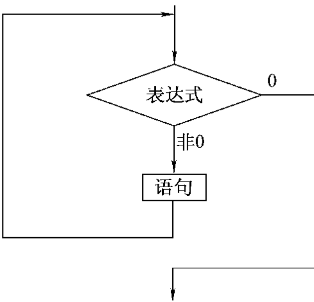


图 4-6　while 语句的执行过程

【例 4-12】　用 while 语句求 $\sum_{n=1}^{100} n$ 。

用传统流程图和 N-S 结构流程图表示算法，如图 4-7 所示。

```
main ( )
{
    int i, sum = 0;
    i = 1;
    while (i < = 100)
          {
            sum = sum + i;
            i + + ;
          }
    printf ( "%d\n", sum);
}
```

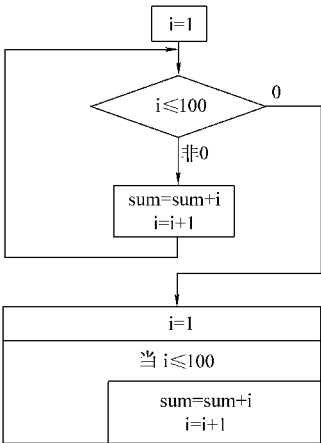


图 4-7　程序流程图

【例 4-13】　编写程序，求 1 ~ 100 累加和。

这是一个求 100 个数的累加和问题。所加的加数从 1 变化到 100，可以看到加数是有规律变化的，后一个加数比前一个加数增 1，第一个加数为 1，最后一个加数为 100。因此，可以在循环中使用一个整型变量 i，每循环一次，使 i 增 1，一直循环到 i 的值超过 100，用这个办法就解决了所需的加数问题。但是要特别注意的是变量 i 需要有一个正确的初值，在这里它的初值应当设定为 0。

下一个要解决的是求累加和。设用一个变量 sum 来存放这 100 个数的和值，可以先求 0 +1 的和并将其放在 sum 中，然后把 sum 中的数加上 2 再放在 sum 中，依次类推。这和人们心算的过程没有什么区别，sum 代表着人们脑中累加的那个和数，不同的是心算的过程由人们自己控制。在这里，sum 累加的过程要放在循环中，由计算机来判断所加的数是否已经超过 100，加数则放在变量 i 中，并在循环过程中一次次增 1。

以下就是求累加和的典型程序。

```
main ( )
```

```
{int i, sum;
i=1;
sum=0;
while (i< =100)
    {sum=sum+i;
i+ +;}
printf ( "sum=%d\ n", sum);
}
```

程序运行后的输出结果;

sum=5050

注意:

1）如果在第一次进入循环时，while 后圆括号内表达式的值为 0，循环一次也不执行。在本程序中，如果 i 的初值大于 100，将使表达式 i< =100 的值为 0，循环体也不执行。

2）在循环体中一定要有使循环趋向结束的操作，以上循环体内的语句 i+ +使 i 不断增 1，当 i>100 时，循环结束．如果没有 i+ +；这一语句，则 a 的值始终不变，循环将无限进行。

3）在循环体中，语句的先后位置必须符合逻辑，否则将会影响运算结果，例如，若将上例中的 while 循环体改写成:

```
while (i< =100)
    {i+ +;
   sum=sum+i;
  }
```

运行后，将输出:

sum=5150

运行的过程中，少加了第一项的值 1，而多加了最后一项的值 101。

【单片机 C 语言设计小实例 1】 程序显示从 1 到 10 的累 加和，读者可以修改一下 while 中的条件看看结果会如果，从而体会一下 while 的使用方法。

```
#include <AT89X51.H>
#include <stdio.h>

void main (void)
{
unsigned int I = 1;
unsigned int SUM = 0; //设初值
SCON = 0x50; //串口方式 1，允许接收
TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数 TH1 = 0xE8; //11.0592MHz 1200 波特率 TL1
 = 0xE8;
TI = 1;
```

```
TR1 = 1; //启动定时器
        while (I< =10)
                {
SUM = I + SUM; //累加
                        printf( "%d SUM=%d\n", I, SUM); //显示
                        I++;
                }
        while (1); //这句是为了不让程序完后, 程序指针继续向下造成程序"跑飞"
}
//最后运行结果是 SUM=55;
```

4.4.4 do-while 语句

do-while 语句的一般形式为

```
do
    语句
while (表达式);
```

1) do 是 C 语言的关键字, 必须和 while 联合使用。

2) do-while 循环由 do 开始, 用 while 结束; 必须注意的是: 在 while (表达式) 后的 ";" 不可丢, 它表示 do-while 语句的结束。

3) while 后一对圆括号中的表达式, 可以是 C 语言中任意合法的表达式, 由它控制循环是否执行。

do-while 语句可以说是 while 语句的补充, while 是先判断条件是否成立再执行循环体, 而 do-while 则是先执行循环体, 再根据条件判断是否要退出循环。这样就决定了循环体无论在任何条件下都会至少被执行一次。其执行过程如图 4-8 所示。

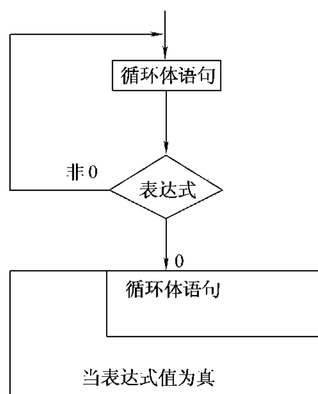


图 4-8 do-while 语句的执行过程

【例 4-14】 求 1 ~ 100 之间的累加和。

程序如下:

```
main ()
{int i, sum=0;
i=1;
do
   {sum=snm+i;
  i++;
  } while (i< =100);
printf ( "%d\n", sum);
```

}

可以看到：对同一个问题可以用 while 语句处理，也可以用 do-while 语句处理。do-while 结构可以转换成 while 结构。

由 do-while 构成的循环与 while 循环十分相似，它们之间的主要区别是：while 循环的控制，出现在循环体之前，只有当 while 后表达式的值为非零时，才可能执行循环体；在 do-while 构成的循环中，总是先执行一次循环体，然后再求表达式的值，因此，无论表达式的值是0 还是非0，循环体至少执行一次。和 while 循环一样，在 do-while 循环体中，一定要有能使 while 后表达式的值变为 0 的操作，否则，循环将会无限制地进行下去。

【例 4-15】 while 和 do-while 循环比较。

（1）
```
main ()
    {int sum =0, i;
    scanf ("%d", &i);
    while (i< =20)
        {sum = sum + i;
      i+ +;
        }
    printf ("sum = %d", sum);
    }
```

（2）
```
main ()
    {int sum =0, i;
    scanf ("%d", &i);
    do
    {sum = sum + i;
    i+ +;
    }
  while (i< =20);
  printf ("sum = %d", sum);
}
```

【单片机 C 语言设计小实例 2】

```
#include <AT89X51. H>
#include  <stdio. h>

void main (void)
{
unsigned int I  =  1;
```

unsigned int SUM = 0；//设初值 SCON = 0x50；//串口方式 1，允许接收 TMOD = 0x20；//定时器 1 定时方式 2

TCON = 0x40；//设定时器 1 开始计数 TH1 = 0xE8；//11.0592MHz 1200 波特率 TL1 = 0xE8；

TI = 1；

TR1 = 1；//启动定时器

do

{

SUM = I + SUM；//累加

printf（“%d SUM = %d\n”，I，SUM）；//显示

I + +；

} while（I < = 10）；while（1）；

}

4.4.5 for 语句

在 C 语言中，for 语句使用最为灵活，它完全可以取代 while 语句。在明确循环次数的情况下，for 语句比以上说的循环语句都要方便简单。它的一般形式为

for（[初值设定表达式]；[循环条件表达式]；[条件更新表达式]）语句

它的执行过程如下：

1）先求解初值设定表达式。

2）求解循环条件表达式，若它的值为真（非 0），则执行 for 语句中指定的内嵌语句，然后执行下面第 3）步；若值为假（0），则结束循环，转到第 5）步。

3）条件更新表达式。

4）转回上面第 2）步继续执行。

5）循环结束，执行 for 语句下面的一个语句。

其执行过程如图 4-9 所示。

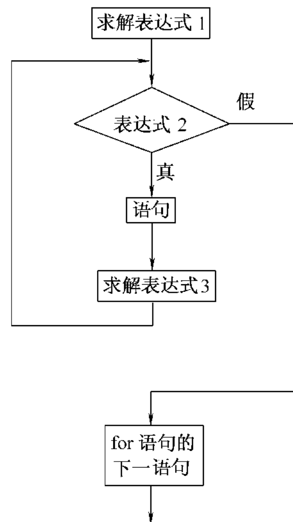


图 4-9　for 语句的执行过程

for 语句最简单的应用形式也是最容易理解的形式如下

for（初值设定表达式；循环条件表达式；条件更新表达式）语句

初值设定表达式总会是一个赋值语句，用来给循环变量赋初值；循环条件表达式是一个关系表达式，由它决定什么条件下执行循环，什么时候退出循环；条件更新表达式，循环变量每循环一次后，都会进行相应的自增或自减运算。这三个部分之间用“；”隔开。

例如：

for（i = 1；i < = 50；i + +）sum = sum + i；

先给变量 i 赋初值 1，判断 i 是否小于或等于 50，如果为真，则执行语句“sum = sum + i”，然后 i 自增 1。然后再重新判断，直到条件为假，即 i > 50 时，循环结束。

相当 while 语句：

```
i = 1;
while (i < = 50)
    { sum = sum + i;
  i + + ;
  }
```

如果将 for 循环语句的表现形式转换成 while 语句形式，则如下所示：

```
初值设定表达式;
while (循环条件表达式)
    {语句
  条件更新表达式;
  }
```

注意：

1）省略了“初值设定表达式”，表示不对循环变量赋初值。

2）省略了“循环条件表达式”，如程序中不做相应的处理，则将变成为死循环。

例如：

```
for (j = 1;; j + + ) sum = sum + j;
```

相当于：

```
j = 1;
while (1)
    {sum = sum + j;
  j + + ;}
```

3）如果不写“条件更新表达式”，则不对循环变量进行操作，这时可在语句体中加入修改循环变量的语句。

例如：

```
for (j = 1; j < = 100;)
    {sum = sum + j;
  j + + ;}
```

4）如果不写“初值设定表达式”和“条件更新表达式”。

例如：

```
for (; j < = 100;)
  {sum = sum + j;
j + + ;}
```

相当于：

```
while (j < = 100)
```

```
{sum = sum + j;
j + + ;}
```

5）for 语句中的 3 个表达式全部省略不写。

例如：

for（;;）语句

相当于：

while（1）语句　即条件永远为真，形成死循环。

6）初值设定表达式可以是设置循环变量初值的赋值表达式，也可以是其他表达式。

7）“初值设定表达式” 和 “条件更新表达式” 可以是一个简单的表达式，也可以是逗号表达式。

for（j =0，i =1；i < =50；i + + ）j =j +i；

或：

for（i =0，j =100；i < =50；i + + ，j--）k =i +j；

8）“循环条件表达式” 一般我们使用关系表达式或逻辑表达式，但也可是数值或字符的表达式，只要其最终的值为非零，即逻辑 “真”，那么就执行循环体。

【单片机 C 语言设计小实例 3】

```
#include < AT89X51. H >
#include  < stdio. h >
void main （void）
{
unsigned int i;
unsigned int SUM  =  0; //设初值 SCON =0x50; //串口方式 1，允许接收 TMOD  =  0x20;
//定时器 1 定时方式 2
TCON =0x40; //设定时器 1 开始计数
TH1  =  0xE8; //11. 0592MHz 1200 波特率 TL1  =  0xE8;
TI =1;
TR1  =  1; //启动定时器
for (i =1; i < =10; i + + ) //这里可以设初始值，所以变量定义时可以不设
    {
        SUM =i +SUM; //累加
        printf（ “% d SUM =% d \ n”，i，SUM）; //显示
    }
while（1）;
}
```

4. 4. 6　循环的嵌套

一个循环体内又完整地包含了另一个循环，称为循环嵌套，前面介绍的 3 种循环都可以互相嵌套，这就是多重循环。但要注意循环层次要清楚，不能交叉。

【例 4-16】 求 2～10000 以内的完全数。

完全数：一个数如果恰好等于它的因子（除开自身）和，该数称为完全数。例如，6 的因子为 1、2、3，而且 1+2+3，因此 6 是“完全数。

程序如下：

```
main ( )
{int i, j, s;
for (i=2; i<=10000; i++)
   {s=0;
  for (j=1; j<i; j++)
    if (i%j==0)
    s+=j;
  if (i==s)
    printf ("%6d\n", s);
  }
}
```

程序包含了一个两重循环 j。其中外层循环变量为 i，控制数据的取值范围、求出因子和，如 i 等于因子和 s，则输出这个数。内层循环变量为 j，内层循环的循环体只有一条语句用于求对应每一个 i 所有的因子和（s）（当 i 能被 j 整除时，j 就是 i 的一个因子）。

几种循环的比较：

前面介绍的各种循环语句，虽然格式不同，但它们有着共同的特点，都实用于循环结构的程序设计。在程序设计的过程中，都具有以下三条内容：

（1）循环体的设计。

（2）循环条件的设计。

（3）循环入口的初始化工作。

从前面所举的几个例子中可以看出，循环体语句的正确执行，依赖于循环的条件，循环的条件依赖循环入口时的初始化工作，一环紧扣一环。

循环体中安排哪些语句，要从分析具体问题入手，前后呼应，合乎逻辑。并且能确保循环能够终止。而且结论正确。

While，do-while 语句的使用，它的循环条件的改变，要靠程序员在循环体中去有意安排某些语句。而 for 语句却不必。使用 for 语句时，若在循环体中想去改变循环控制变量，以期改变循环条件，无异于画蛇添足。

While 循环，do-while 循环适用于未知循环的次数的场合，而 for 循环适用于已知循环次数的场合。使用哪一种循环又依具体的情况而定。

凡是能用 for 循环的场合，都能用 While，do-while 循环实现，反之则未必。

4.4.7 break 和 continue 语句

1. break 语句

break 语句通常用在循环语句（for，while，do-while）和 switch 语句中。当 break 用于

switch 语句中时，可使程序跳出 switch 程序体而执行 switch 语句后面的语句；如果没有 break 语句，则将成为一个死循环。break 在 switch 中的用法非常简单，加上一句“break;”即可。

当 break 语句用于 do-while、for、while 循环语句中时，可使程序终止循环，跳出循环体而执行后面的语句，通常 break 语句与 if 语句一起使用，当满足条件时便跳出循环。

【例 4-17】 计算半径 r = 1 ~ 10 时的圆面积，直到面积 area 大于 100 为止。

程序如下：

```
    #define PI   3.14159
main ( )
{
  float r, area;
  for ( r=1; r<=10; r++ )
    {
      area=PI*r*r ;
      if (area>100) break;
      printf ( "%f", area);
    }
}
```

从上面的 for 循环可以看到当 area > 100 时，执行 break 语句，提前终止执行循环，即不再继续执行其余的几次循环。

2. continue 语句

continue 语句是用于中断的语句，通常使用在循环中，它的作用是结束本次循环，跳过循环体中还没有执行到的语句，而跳转到下一次循环周期。

语法为：continue;

continue 同时也是一个无条件跳转语句，但功能和前面说到的 break 语句有所不同，continue 执行后不是跳出循环体，而是跳到循环的开始部分并执行下一次的循环。

continue 语句只用在 for、while、do-while 等循环体中，通常与 if 语句一起使用。

Continue 语句与 break 语句的区别及其执行过程如图 4-10 所示。

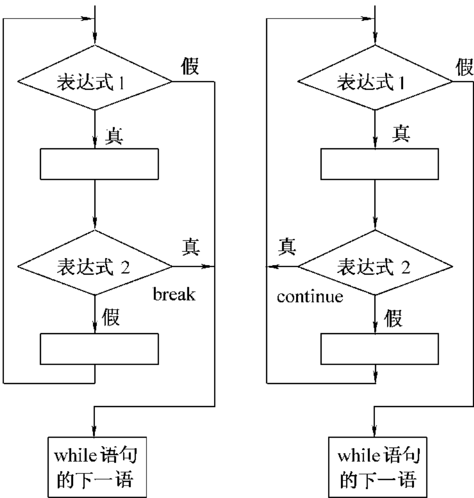


图 4-10 break，continue 语句的执行过程比较

1）while（表达式 1）

```
    { ……
    if (表达式 2) break;
    ……
```

```
}
```

2）while（表达式 1）

```
    { ……
      if（表达式 2） continue;
      ……
    }
```

【例 4-18】

```
main ( )
{
int i;
for (i = 1; i < = 20; i + + )
   {
   if (i%2 = = 0) continue;
   printf ( "%d", i);
   }
}
```

这个程序是将 1～20 之间不能被 2 整除的数输出到屏幕上。当变量 i 对 2 取模为 0 时，则为 i 能整除 2，此时执行 continue 语句，程序将跳到 for 语句继续执行，输出函数在此时将不会被执行。这个程序是针对 for 语句的。如果在其他两种循环中使用了 continue 语句，程序将跳过剩余的循环体内的语句，直接转到条件表达式处开始判断表达式是否成立，然后继续执行程序。

3. 程序举例

【例 4-19】 用公式 $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 求 π。

N-S 流程如图 4-11 所示。

```
#include < math. h >
main ( )
{
   int s;
   float n, t, pi;
   t = 1, pi = 0; n = 1.0; s = 1;
   while (fabs (t)  > 1e-6)
      {pi = pi + t;
      n = n + 2;
      s = - s;
```

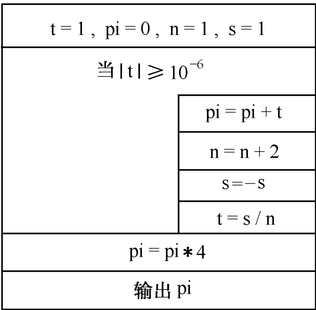


图 4-11　N-S 流程图

```
        t = s/n;
        }
pi = pi * 4;
printf ("pi = %10.6f\n", pi);
}
```

【例 4-20】 判断 m 是否为素数。

N-S 流程如图 4-12 所示

```
#include <math.h>
main ()
{
int m, i, k;
scanf ("%d", &m);
k = sqrt (m);
for (i = 2; i <= k; i++)
   if (m%i == 0) break;
   if (i >= k + 1)
   printf ("%d is a prime number\n", m);
   else
   printf ("%d is not a prime number\n", m);
}
```

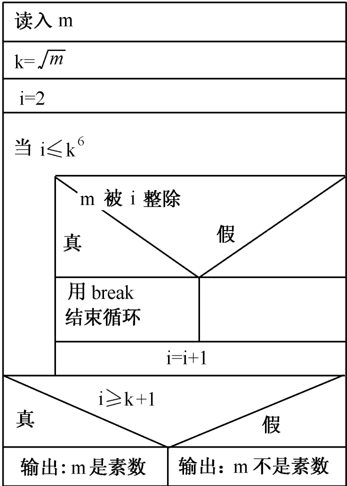


图 4-12 N-S 流程图

【例 4-21】 求 100～200 间的全部素数。

```
#include <math.h>
main ()
{
   int m, i, k, n = 0;
   for (m = 101; m <= 200; m = m + 2)
       {
        k = sqrt (m);
        for (i = 2; i <= k; i++)
           if (m%i == 0) break;
           if (i >= k + 1)
                 {printf ("%d", m);
               n = n + 1;}
           if (n%n == 0) printf ("\n");
       }
printf ("\n");
}
```

第 5 章　编译预处理与位运算预处理命令

5.1　概述

ANSI C 标准规定，C 源程序中可以加入一些预处理命令，以改进程序设计环境，提高编程效率。所谓预处理是指在进行编译的第一遍扫描之前所作的工作，它由预处理程序负责完成。当对一个源文件进行编译时，系统将自动引用预处理程序对源程序中的预处理部分进行处理，处理完毕后再进行编译工作。预处理命令不是 C 语言本身的组成部分，所以在使用时以“#”开头，以示和 C 语言的区别。在前面的章节中，我们已看到过多次以“#”开头的预处理命令。

常用的预处理功能有：宏定义、文件包含、条件编译。

5.2　宏定义

C 语言允许用一个标识符来表示一个字符串，称为“宏”。被定义为“宏”的标识符称为“宏名”。在编译预处理时，对程序中所有的“宏名”，都用宏定义中的字符串去代替，称为“宏代换”。

宏定义由宏定义命令完成；宏代换由预处理程序完成。

在 C 语言中，“宏”定义分为两种：不带参数的宏定义和带参数的宏定义。

5.2.1　不带参数的宏定义

定义的一般形式为

```
#define　标识符　字符串
```

其含义是出现标识符的地方均用字符串来替代。

如：#define PI 3.1415926

作用：用标识符（称为“宏名”）PI 代替字符串“3.1415926”。

宏展开：用定义的字符串去替换标识符，然后再对替换处理后的源程序进行编译，这一过程称为宏展开。在预编译时，将源程序中出现的宏名 PI 替换为字符串“3.1415926”，这一替换过程称为“宏展开”。

#define：宏定义命令

#undef：终止宏定义命令

【例 5-1】

```
#define PI 3.1415926
main ( )
```

```
{ float 1, s, r, v;
printf ( "input radius:");
scanf ( "%f", &r); /* 输入圆的半径 */
1 = 2.0 * PI * r; /* 圆周长 */
s = PI * r * r; /* 圆面积 */
v = 4.0/3.0 * PI * r * r * r; /* 球体积 */
printf ( "1 =%10.4f\ ns =%10.4f\ nv =%10.4f\ n", 1, s, v);
}
```

注意：

1）宏定义必须以#define 开头，行末不加分号；

2）#define 命令一般出现在函数外部；

3）每一个#define 只能定义一个宏，且只占一行；

4）宏定义中的宏体只是一串字符，没有值和类型的含义，编译系统只对程序中出现的宏名用定义中的宏体作简单替换，而不作语法检查，且不分配内存空间；

5）宏体为空时，宏名被定义为字符常量 0。

关于宏定义的说明：

1）一般宏名用大写字母表示。(变量名一般用小写字母)。

2）使用宏可以提高程序的可读性和可移植性。如上述程序中，多处需要使用 π 值，用宏名既便于修改又意义明确。

3）宏定义是用宏名代替字符串，宏扩展时仅作简单替换，不检查语法。语法检查在编译时进行。

4）宏定义不是 C 语句，后面不能有分号。如果加入分号，则连分号一起替换。

如：

```
#define PI 3.1415926;
area = PI * r * r;
```

在宏扩展后成为：

```
area = 3.1315926; * r * r;
```

结果，在编译时出现语法错误。

5）通常把#define 命令放在一个文件的开头，使其在本文件全部有效。(#define 定义的宏仅在本文件有效，在其他文件中无效，这与全局变量不同)。

6）宏定义终止命令 #undef 结束先前定义的宏名。

```
#define G 9.8
main ( )
{
}
#undef G /* 取消 G 的意义 */
f1 ( )
```

¦

7）宏定义中可以引用已定义的宏名。

【例 5-2】

```
#define R 3.0
#define PI 3.1415926
#deinfe L 2 * PI * R
#define S PI * R * R
main ( )
{
printf ( "L = %f \ nS = %f \ n", L, S);
}
```

8）对程序中用双引号括起来的字符串，即使与宏名相同，也不替换。例如上例的 printf 语句中，双引号括起来的 L 和 S 不被替换。

5.2.2 带参数的宏定义

定义的一般形式为

#define 宏名（参数表）字符串

其含义是作相应的参数替换，带参数的宏在展开时，不是进行简单的字符串替换，而是进行参数替换。如图 5-1 所示。

【例 5-3】

```
#define PI 3.1415926
#define S (r) PI * r * r
main ( )
{ float a, area;
a = 3.6;
area = S (a);
printf ( "r = %f \ narea = %f \ n", a, area);
}
```

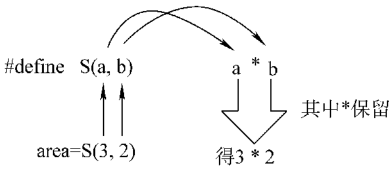


图 5-1 define 定义

说明：

1）带参数的宏展开时，用实参字符串替换形参字符串，注意可能发生的错误。比较好的办法是宏定义的形参加括号。见表 5-1

表 5-1 宏定义展开

宏定义	语句	展开后	
#define S (r) PI * r * r	area = S (a + b);	area = PI * a + b * a + b;	✗
#define S (r) PI * (r) * (r)	area = S (a + b);	area = PI * (a + b) * (a + b)	

2）宏定义时，宏名与参数表间不能有空格。例如：#define S□（r）PI＊r＊r（□表示空格）

带参数的宏定义与函数的区别，见表 5-2。

表 5-2　带参数的宏定义与函数的区别

	函　　数	宏
信息传递	实参的值或地址传送给形参	用实参的字符串替换形参
处理时刻及内存分配	配程序运行时处理，分配临时内存单元	宏展开在预编译时处理，不存在分配内存的问题
参数类型	实参和形参类型一致。如不一致，编译器进行类型转换	字符串替换，不存在参数类型问题
返回值	可以有一个返回值	可以有多个返回值，［例 5-4］
对源程序的影响	无影响	宏展开后使程序加长
时间占用	占用程序运行时间	占用编译时间

【例 5-4】　返回多个值的宏定义。

```
#define PI 3. 1415926
#define CIRCLE（R，L，S，V）L =2 ＊PI ＊R；S =PI ＊R ＊R；V =4/3 ＊PI ＊R ＊R ＊R
main（）
{ float r，l，s，v；/＊半径、圆周长、圆面积、球体积 ＊/
scanf（“%f”，&r）；
CIRCLE（r，l，s，v）；
printf（“r =%6. 2f，l =%6. 2f，s =%6. 2f，v =%6. 2f\ n”，r，l，s，v）；
}
```

【例 5-5】　输出格式定义为宏

```
#define PR printf
#define NL“ \ n”
#define D“%d ”
#define D1 D NL
#define D2 D D NL
#define D3 D D D NL
#define D4 D D D D NL
#define S“%s”

main（）
{ int a，b，c，d；
char string［］ =“CHINA”；
a = 1；b = 2；c = 3；d = 4；
```

```
PR（D1，a）;
PR（D2，a，b）;
PR（D3，a，b，c）;
PR（D4，a，b，c，d）;
PR（S，string）;
}
```

5.3 文件包含

文件包含是指将一个源文件的全部内容包含到另一个源文件中去，成为其一部分。文件包含预处理命令的一般格式为

#include ＜文件名＞ 或

#include“文件名”

两种格式的区别在于：用＜＞括起文件名的文件包含命令，指示系统只在指定存放头文件的目录下（include 子目录下）查找该文件。而用“”的命令，系统首先在源文件所在目录下查找该文件，如果找不到，再到指定存放头文件的目录下查找该文件。这里所说的“头文件”，是因为#include 命令所指定的被包含文件常放在文件的开头，习惯上称被包含文件为头文件，并常以 h 作为其文件的扩展名，如“stdio. h”。用“”的文件包含命令中的双引号中的文件名还可改为文件路径，如“c：\ tc \ include \ stdio. h”。

用#include 文件包含预处理命令的好处是：当许多程序中需要用到一些共同的常量、数据等资料时，可以把这些共同的东西写在以 . h 作为扩展名的头文件中，如哪个程序需要用就可用文件包含命令把它们包含进来，省去了重复定义的麻烦。

例如：有以下文件 f. c

```
#include ＜stdio. h＞
#define PI 3. 1415926
#define AREA（r）(PI＊（r）＊（r))
#define PR printf
#define D“%f”
```

下面程序要用到以上内容，就可用文件包含命令把它们包含进来，形成一个新的源程序。

```
#include“c：\ tc \ f. c”
main（）
{ float r＝3. 5，s;
s＝AREA（3. 5）;
PR（D，s）;}
```

一个包含命令只能包含一个文件，若要包含 n 个文件，就需要 n 个包含命令。

5.4 条件编译

条件编译：一般情况下，源程序中的所有代码均参加编译，但有时只希望部分代码进行编译，称为“条件编译”。

条件编译命令可以分为以下几种：

1. #ifdef

#ifdef 标识符 程序段 1 #else 程序段 2 #endif	#ifdef 标识符 程序段 1 #endif

其中，“标识符”用#define 命令定义。

【例 5-6】 程序调试信息的显示。

```
#define DEBUG
#ifdef DEBUG
printf（“x=%d，y=%d，z=%d\ n”，x，y，z）；
#endif
```

printf（）语句被编译，程序运行时可以显示 x、y、z 的值。在程序调试完成后，不再需要显示 x、y、z 的值，则只需要去掉 DEBUG 标识符的定义。

直接使用 printf（）语句显示调试信息，在程序调试完成后去掉 printf（）语句，也可以达到目的。但如果程序中有很多处需要调试观察，增、删语句既麻烦又容易出错。而使用条件编译则相当清晰、方便。

2. #ifndef

#ifndef 标识符 程序段 1 #else 程序段 2 #endif	#ifndef 标识符 程序段 1 #endif

3. #if

#if 表达式
程序段 1

```
#else
程序段 2
#endif
```

【例 5-7】 输入一行字母字符，根据需要设置条件编译，使之能将字母全改为大写输出，或全改为小写输出。

```
#define LETTER 1
main ( )
{
char str [20] = “C Language”, c;
int i;
i = 0;
while ( (c = str [i]) ! = ‘ \ 0’)
{ i + +;
#if LETTER
if (c > = ‘a’ && c < = ‘z’)
c = c - 32;
#else
if (c > = ‘A’ && c < = ‘Z’)
c = c + 32;
#endif
printf ( “% c”, c);
}
}
```

【例 5-8】

```
#define MAX 80
main ( )
{int i = 10; float x = 25. 8;
#if MAX >99
printf ( “% d \ n”, i);
#else
printf ( “% f \ n”, x);
#endif
printf ( “% d,% f \ n”, i, x);
}
```

运行结果 10

10，25.800000
将 MAX 定义为 80，执行 printf（“%f\n”，x）；
运行结果 25.800000
10，25.800000

5.5 位操作运算符

前面介绍的各种运算都是以字节作为最基本位进行的。但在很多系统程序中常要求在位（bit）一级进行运算或处理。C 语言提供了位运算的功能，这使得 C 语言也能像汇编语言一样用来编写系统程序。

利用位操作运算符可对一个数按二进制格式进行位操作。

1. 按位“与”运算 &

b3 = b1&b2
例如：char b1 = 25，b2 = 77，化成二进制（或十六进制）表示为

b1 = 00011001（或 0x1B）
b2 = 01001101（或 0x4D）
b3 = 00001001（或 0x09）即 b3 = 9
& 可用作“掩码”运算，即屏蔽掉某些位。例如，掩码为 127，化成二进制表示为 01111111，可屏蔽掉第 7 位。

2. 按位“或”运算 |

b3 = b1 | b2
b1 = 00011001（或 0x1B）
b2 = 01001101（或 0x4D）
b3 = 01011101（或 0x5D）即 b3 = 93

3. 按位“异或”运算^

b3 = b1^b3
b1 = 00011001（0x1B）
b2 = 01001101（0x4D）
b3 = 01010100（0x54）即 b3 = 84

很容易验证：b3 = b1^b2^b2 = b1。这个特性可用于某些数据处理。例如，一个文本的每个字经过与一个关键字做异或处理，就被简单地加密了。要解密时，只需用同一个关键字再做一次异或处理，使其恢复原样。又如，用下列异或处理程序：

```
swap1（a，b）
int a，b；
```

```
{
a = a^b;
b = a^b;
a = a^b;
}
```

代替下列程序：

```
swap (a, b)
int a, b;
{
int temp;
temp = a;
a = b;
b = temp;
}
```

同样实现 a 和 b 交换，异或处理的速度更快。

4. 按位取反运算符 ~

w2 = ~w1，w2 为 w1 二进制按位取反。

例如：unsigned int w1 =0122457，w2；

二进制表示（8 进制表示）

w1 1010010100101111（0122457）

w2 0101101011010000（0055320）

5. 左移运算符 <<

左移运算符“ << ”是双目运算符。其功能把“ << ”左边的运算数的各二进位全部左移若干位，由“ << ”右边的数指定移动的位数，高位丢弃，低位补 0。

例如：

a << 4

指把 a 的各二进位向左移动 4 位。如 a =00000011（十进制 3），左移 4 位后为 00110000（十进制 48）。

6. 右移运算符 >>

右移运算符“ >> ”是双目运算符。其功能是把“ >> ”左边的运算数的各二进位全部右移若干位，由“ >> ”右边的数指定移动的位数。

例如：

设 a =15，

a >> 2

表示把 000001111 右移为 00000011（十进制 3）。

应该说明的是，对于有符号数，在右移时，符号位将随同移动。当为正数时，最高位补 0，而为负数时，符号位为 1，最高位是补 0 或是补 1 取决于编译系统的规定。Turbo C 和很多系统规定为补 1。

【例 5-9】

```
main ( ) {
    unsigned a, b;
    printf ( "input a number: ");
    scanf ( "%d", &a);
    b = a >> 5;
    b = b&15;
    printf ( "a = %d \ tb = %d \ n", a, b);
}
```

【例 5-10】

```
main ( ) {
    char a = 'a', b = 'b';
    int p, c, d;
    p = a;
    p = (p << 8) | b;
    d = p&0xff;
    c = (p&0xff00) >> 8;
    printf ( "a = %d \ nb = %d \ nc = %d \ nd = %d \ n", a, b, c, d);
}
```

第 6 章　数组与函数

数组是一组具有相同类型和名称的变量的集合。这些变量称为数组的元素，每个数组元素都有一个编号，这个编号叫做下标，我们可以通过下标来区别这些元素。数组元素的个数有时也称之为数组的长度。在 C 语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。

6.1　一维数组的定义和引用

6.1.1　一维数组的定义方式

举个形象点的例子，就像学校操场上队列，每一个年级代表一个数据类型，每一个班级为一个数组，每一个学生就是数组中的一个元素。数组中的每个数据都可以用唯一的下标来确定其所在位置，下标可以是一维或多维的。比如在学校的方队中要找一个学生，这个学生在 I 年级 H 班 X 组 Y 号，那么可以把这个学生看作在 I 类型的 H 数组中（X，Y）下标位置中。数组和普通变量一样，要求先定义再使用，下面是定义一维或多维数组的方式：

一维数组的定义方式为

```
类型说明符 数组名［常量表达式］;
```

其中，“类型说明符”是指数组中的各数据单元的类型，每个数组中的数据单元只能是同一数据类型。“数组名”是整个数组的标识，命名方法和变量命名方法是一样的。在编译时系统会根据数组大小和类型为变量分配空间，数组名可以说就是所分配空间首地址的标识。“常量表达式”是表示数组的长度和维数，它必须用“［ ］”括起，方括号里的数不能是变量只能是常量。

例如：

int a［5］;　　　　　　说明整型数组 a，有 5 个元素。
float b［8］，c［8］;　说明实型数组 b 和实型数组 c，各有 8 个元素。
char ch［10］;　　　　说明字符数组 ch，有 10 个元素。

在 C 语言中数组的下标是从 0 开始的，而不像有些编程语言那样是从 1 开始的。如一个具有 8 个数据单元的数组 a，它的下标就是从 a［0］～a［7］，如引用单个元素就是数组名加下标，如 a［1］就是引用 a 数组中的第 2 个元素，如果错用了 a［8］就会出现错误。还有一点要注意的就是在程序中只有字符型的数组可以一次引用整个数组，其他类型的，则需要逐个引用数组中的元素，不能一次引用整个数组。

对于数组类型说明应注意以下几点：

1）数组的类型根据数组元素的数据类型所定。对于同一个数组，其所有元素的数据类型都是相同的。

2）数组名不能和其他变量名相同。

例如：

```
main ( )
{
  float a;
  int a [9];
  ……
}
```

这样的定义是错误的。

3）方括号中常量表达式表示数组元素的个数，如 a [8] 表示数组 a 有 8 个元素。8 个元素分别为 a [0]，a [1]，a [2]，a [3]，a [4] ，a [5] ，a [6] ，a [7]。

4）方括号中的变量表达式不可以是变量，但可以是符号常数或常量表达式。

例如：

```
#define AA 6
main ( )
{
float a [1 +8], b [7 + AA];
……
}
```

这样的写法是合法的。

但是下面这样的写法是错误的。

```
main ( )
{
int m =8;
int a [m];
……
}
```

5）允许在同一个类型说明中，说明多个数组和多个变量。

例如：

```
int aa, bb, cc, dd, m [10], n [20];
```

6.1.2 一维数组元素的引用

当定义了一个数组，则数组中的各个元素就共用一个数组名（ 即该数组变量名），它们之间是通过下标不同以示区别的。对数组的操作归根到底就是对数组元素的操作。

数组元素的一般形式为：

数组名［下标］

其中下标只能为整型常量或整型表达式。如为小数时，C 编译器将自动取整。

例如：

```
a［8］
a［i+j］
a［j++］
```

这些都是合法的数组元素。

数组元素通常也称为下标变量。必须先定义再使用下标变量。下标变量的值不允许超越所定义的下标下界和上界。

例如，一个数组有 20 个元素，我们将其各元素值逐个输出：

```
for（i=0；i<20；i++）
    printf（“%d”，a［i］）；
```

如改成下面的写法，那会出现错误的，因为 C 语言中不能用一个语句输出整个数组。

```
printf（“%d”，a）；
```

【例 6-1】

```
main（）
{
  int i，a［10］；
  for（i=0；i<=9；i++）
    a［i］=i；
  for（i=9；i>=0；i--）
    printf（“%d ”，a［i］）；
}
```

【例 6-2】

```
main（）
{
  int i，a［10］；
  for（i=0；i<10；）
    a［i++］=i；
  for（i=9；i>=0；i--）
    printf（“%d”，a［i］）；
}
```

【例 6-3】

```
main（）
{
```

```
    int i, a [10];
    for (i=0; i<10;)
      a [i++] =2*i+1;
    for (i=0; i<=9; i++)
      printf ("%d", a [i]);
    printf ("\n%d %d\n", a [5.2], a [5.8]);
}
```

本例中用一个循环语句给数组 a 各元素送入奇数数字，然后在第二个循环语句中，一一输出各个奇数。在第一个 for 语句中，表达式 3 省略了，因为在下标变量中使用了表达式 i++，用来修改循环变量的值。当然第二个 for 语句也可以这样写。程序最后面一个 printf 语句输出了两次 a［5］的值，因为当下标不为整数时编译器将自动取整。

6.1.3 一维数组的初始化

数组元素赋值的 3 种方法：数组定义后，数组元素的值是随机的，可以用三种方法给数组元素赋值。

1）用赋值语句给数组元素赋值。

2）用输入函数从键盘或数据文件中读取数据并赋给数组元素。

3）定义数组的同时，为数组元素赋值。这种方法就是我们所说的数组初始化。

前两种方法是在程序的运行阶段实行的，每运行一次程序，相关的语句都必须执行一遍，占用程序执行时间较多，后 1 种方法对于静态存储的数组是在程序编译阶段完成赋初值的，只要程序编译成功，运行程序时不再执行相关的语句，减少了程序运行时间。另外，前两种方法对没有赋值的数组元素来说，数组元素的值是不确定的，第 3 种方法系统对没有赋初值的数组元素，自动赋予一个确定值（整型或实型数组元素赋数值 0，字符型数组元素赋字符常量‘\0’）。

初始化赋值的一般形式为

类型说明符 数组名［常量表达式］ = {初值，初值...... 初值}；

其中在 { } 中的各个数据为各数组元素的初值，各值之间用逗号分隔。

例如：

```
int a [10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

相当于 a［0］ =0；a［1］ =1...a［9］ =9；

对数组元素的初始化可以用以下方法实现：

1）在定义数组时对数组元素赋初值。例如：

```
int a [10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

2）可以只给一部分元素赋值。例如：

```
int a [10] = {0, 1, 2, 3, 4};
```

3）对全部元素赋初值时，可以不指定数组长度。例如：

int a［5］ =｛0，1，2，3，4｝；

可以写成

int a［ ］ =｛0，1，2，3，4｝；

6.1.4 一维数组程序举例

可以在程序执行过程中，对数组作动态赋值。这时可用循环语句配合 scanf 函数逐个对数组元素赋值。

【例 6-4】

```
main（）
{
  int i，max，a［10］；
  printf（“input 10 numbers：\ n”）；
  for（i=0；i<10；i++）
    scanf（“%d”，&a［i］）；
  max=a［0］；
  for（i=1；i<10；i++）
    if（a［i］ >max）max=a［i］；
  printf（“maxmum=%d\ n”，max）；
}
```

本例程序中第一个 for 语句逐个输入 10 个数到数组 a 中。然后把 a［0］送入 max 中。在第二个 for 语句中，从 a［1］到 a［9］逐个与 max 中的内容比较，若比 max 的值大，则把该下标变量送入 max 中，因此 max 总是在已比较过的下标变量中的最大者。比较结束，输出 max 的值。

【例 6-5】

```
main（）
{
  int i，j，p，q，s，a［10］；
  printf（“\ n input 10 numbers：\ n”）；
  for（i=0；i<10；i++）
    scanf（“%d”，&a［i］）；
  for（i=0；i<10；i++）{
    p=i；q=a［i］；
    for（j=i+1；j<10；j++）
    if（q<a［j］）{ p=j；q=a［j］；}
    if（i！ =p）
```

```
        {s = a[i];
        a[i] = a[p];
        a[p] = s;}
      printf("%d", a[i]);
   }
}
```

本例程序中用了两个并列的 for 循环语句，在第二个 for 语句中又嵌套了一个循环语句。第一个 for 语句用于输入 10 个元素的初值。第二个 for 语句用于排序。本程序的排序采用逐个比较的方法进行。在 i 次循环时，把第一个元素的下标 i 赋于 p，而把该下标变量值 a［i］赋于 q。然后进入小循环，从 a［i+1］起到最后一个元素止逐个与 a［i］作比较，有比 a［i］大者则将其下标送 p，元素值送 q。一次循环结束后，p 即为最大元素的下标，q 则为该元素值。若此时 i≠p，说明 p，q 值均已不是进入小循环之前所赋之值，则交换 a［i］和 a［p］之值。此时 a［i］为已排序完毕的元素。输出该值之后转入下一次循环，对 i+1 以后各个元素排序。

6.2　二维数组的定义和引用

6.2.1　二维数组的定义

前面我们介绍了一维数组，它只有一个下标，其数组元素也称为单下标变量。

生活中，有很多事物，仅仅用一维数组，将无法恰当地被表示。还是说学生管理吧。假如，一个班级 40 个学员，把他们编成 1～40 号。但现在有两个班级要管理怎么办？每个班级都各有各的编号，比如 1 班学生的编号是 1～40；2 班的学生也是 1～40。现在两个班的学生编号要混在一起输入计算机系统，从 1 号编到 80 号，显然不是很合适。在实际问题中有很多量是二维的或多维的，因此 C 语言允许构造多维数组。多维数组元素有多个下标，以标识它在数组中的位置。在此，我们主要介绍一下二维数组，类似的还有三维数组，我们在此就不讲了。

二维数组定义的一般形式是

类型说明符 数组名［常量表达式 1］［常量表达式 2］

其中常量表达式 1 表示第一维大小，常量表达式 2 表示第二维大小。

例如：

```
int a[4][5];
```

说明了一个 4 行 5 列的数组，数组名为 a，其下标变量的类型为整型。该数组的数组元素共有 4×5 个，即：

a［0］［0］，a［0］［1］，a［0］［2］，a［0］［3］，a［0］［4］
a［1］［0］，a［1］［1］，a［1］［2］，a［1］［3］，a［1］［4］
a［2］［0］，a［2］［1］，a［2］［2］，a［2］［3］，a［2］［4］

a［3］［0］, a［3］［1］, a［3］［2］, a［3］［3］, a［3］［4］

由于数组 a 为 int 类型，int 类型数据占两个字节的内存空间，所以数组中每个元素均占有两个字节。

6.2.2 二维数组元素的引用

二维数组的表示的形式为

数组名［下标］［下标］

其中下标应为整型常量或整型表达式。

例如：

a［4］［5］

表示 a 数组有 4 行 5 列元素。

下标变量和数组定义时的元素个数有些相似，但这两者具有不同的含义。数组定义时方括号中给出的是某一维的长度，即长度的最大值；而数组元素中的下标变量是该元素在数组中的位置标识。注意：前者只能是常量，后者可以是常量，变量或表达式。

【例 6-6】 一个学习小组有 4 个人，每个人有三门课的考试成绩。求全组分科的平均成绩和各科总平均成绩。见表 6-1。

表 6-1 学生成绩分布表

	张	王	李	赵
语文	80	61	59	85
数学	75	65	63	87
英语	92	71	70	90

可设一个二维数组 a［4］［3］存放 4 个人三门课的成绩。再设一个一维数组 v［3］存放所求得各分科平均成绩，设变量 average 为全组各科总平均成绩。编程如下：

```
main ()
{
  int i, j, s=0, average, v[3], a[4][3];
  printf ("input score\n");
  for (i=0; i<3; i++)
   {
    for (j=0; j<4; j++)
    { scanf ("%d", &a[j][i]);
      s=s+a[j][i];}
    v[i] =s/4;
    s=0;
```

```
    }
    average = (v[0] + v[1] + v[2])/3;
    printf("语文:%d\n数学:%d\n英语:%d\n", v[0], v[1], v[2]);
    printf("平均分:%d\n", average);
}
```

程序中首先用了一个双重循环。在内循环中依次读入某一门课程的各个学生的成绩，如第一次循环时，读入每个人的语文成绩，同时把所有人的语文成绩累加起来，退出内循环后再把该累加成绩除以 4 送入 v［i］之中，这就是该门课程的平均成绩。因为我们总共统计三门课程，所以外循环共循环三次，分别求出三门课各自的平均成绩并存放在 v 数组之中。退出外循环之后，把 v［0］，v［1］，v［2］相加除以 3 即得到各科总平均成绩。最后按题意输出各个成绩。

6.2.3 二维数组的初始化

二维数组初始化可以在数组定义时，给各下标变量赋以初值。二维数组即可分段赋值，又可按行连续赋值。

例如对数组 a［4］［3］：

1）按分段进行赋值为

int a［4］［3］ = {{80，75，92}，{61，65，71}，{59，63，70}，{85，87，90}}；

2）按连续进行赋值为

int a［4］［3］ = {80，75，92，61，65，71，59，63，70，85，87，90}；

其最终效果都是一样的。

对于二维数组初始化赋值还要注意几点：

1）可以只对部分元素赋初值，未赋初值的元素自动取 0 值。

例如：

int a［2］［2］ = {{1}，{2}}；

是对第 1 行第 1 列和第 2 行第 1 列元素赋值，其他元素为 0。语句执行后各元素的值为：

1 0 0

2 0 0

int a［3］［3］ = {{0，5}，{0，0，7}，{1}}；

赋值后的元素值为：

0 5 0

0 0 7

1 0 0

2）如要对所有数组元素赋初值，那么定义时，第一维的长度可以不写。

例如：

int a［3］［3］ = {9，8，7，6，5，4，3，2，1}；

可以改为：

int a［］［3］ = {9，8，7，6，5，4，3，2，1}；

6.3　字符数组

用来存放字符型数据的数组我们称为字符数组。

6.3.1　字符数组的定义

前面的例子中我们用整型表示成绩，所以数组是一个整型数组。其实不同类型的数组定义形式都基本类似，其他的数据类型，如浮点型 float、布尔型 bool 或者字符型 char 均可以有相关的数组，如：

int age［］{25，32，29，40}；

float money［］ = {1000.50，2000，2870.95}；

bool married［］ = {false，true}；

char name［］ = {‘M’，‘i’，‘k’，‘e’}；

最后一行定义了一个字符数组，用来存储一个英文人名：“Mike”。

字符数组也可以是二维或多维数组。

例如：

char c［4］［3］；

即为二维字符数组。

6.3.2　字符数组的初始化

与整型数组一样，字符数组也可以在定义时作初始化赋值。

例如：

char a［8］ = {‘p’，‘r’，‘o’，‘g’，‘r’，‘a’，‘m’}；

赋值后各元素的值为：

c［0］的值为‘p’
c［1］的值为‘r’
c［2］的值为‘o’
c［3］的值为‘g’
c［4］的值为‘r’
c［5］的值为‘a’
c［6］的值为‘m’

当对全体元素赋初值时也可以省去长度说明。c［7］为‘\0’，作为字符串结束的标志。

例如：

char c［］ = {‘p’，‘r’，‘o’，‘g’，‘r’，‘a’，‘m’}；

这时 C 数组的长度自动定为 9。

6.3.3 字符数组的引用

【例 6-7】

```
main ( )
{
  int i, j;
  char a [ ] [5] = {{'C', 'h', 'i', 'n', 'a',}, {'J', 'a', 'p', 'a', 'n'}};
  for (i =0; i< =1; i+ +)
     {
       for (j =0; j< =4; j+ +)
          printf ( "%c", a [i] [j]);
       printf ( "\ n");
     }
}
```

例 6-7 中二维字符数组 a 在初始化时将全部元素赋以初值，因此一维下标的长度可以不写，然后通过循环语句将各字符一一输出。

6.3.4 字符串和字符串结束标志

在 C 语言中不存在字符串变量，通常用一个字符数组来存放一个字符串。前面我们也曾提到过字符串总是以‘\0’作为串的结束标记。由此，在把一个字符串存入数组时，也把结束符‘\0’存入了数组，以此作为该字符串结束的标志。

除了定义字符数组时初始化各元素值，还可以用字符串的形式来进行赋值。

例如：

char c [ ] = {‘p’，‘r’，‘o’，‘g’，‘r’，‘a’，‘m’}；

可写为

char c [ ] = {“C program”}；或：char c [ ] = “C program”；

用字符串方式赋值比用字符逐个赋值要多占 1 个字节，用于存放字符串结束标志‘\0’。上面的数组 c 在内存中的实际存放情况为

p	r	o	g	r	a	m	\0

‘\0’是程序在编译时，由编译器自动加上的。由于有了‘\0’标志，所以在用字符串赋初值时一般无须指定数组的长度，而由系统自行处理。

6.4 函数概述

在前面已经介绍过，C 程序是由函数组成的。对于规模较大，比较复杂的问题，人们常采用模块化设计方法。即将一个较大的程序按功能划分成若干个程序模块，每一个模块用来实现一个特定的功能。在 C 语言中，函数就是实现模块化程序设计的工具。C 语言中的函数

相当于其他高级语言中的子程序和过程。C 语言系统本身提供了极为丰富的库函数，还允许用户建立自定义函数。用户可把自己的程序写成一个一个相对独立的函数，然后在需要用到它的地方来调用它。C 语言可谓是函数的集合。

由于采用了函数结构的写法，使 C 语言的程序代码结构清晰，同时有利于程序的编写、阅读和维护。

6.4.1 函数定义的一般形式

1. 无参函数的定义形式

```
类型标识符 函数名（）
{声明部分
语句
}
```

其中类型标识符指明了函数值的类型，也可以说是函数返回值的类型。函数名是由用户自己定义，函数名后面的一个空括号，因为没有函数参数，所以里面没有内容，但括号不可以省略不写。

花括号中的内容为函数体。在函数体中声明的各种对象，在该函数体内有效。

由于大多数情况下，无参函数没有返回值，此时我们可以将函数类型符写成 void。

我们可以改写一个函数定义：

```
void print( )
{
  printf (“This is a example \n”);
}
```

从这个例子中，我们可以看出，print 为函数名，并且它是一个无参函数，当被其他函数调用时，它的功能是输出一个 This is a example 字符串。

2. 有参函数的定义形式

```
类型标识符 函数名（形式参数列表）
{声明部分
语句
}
```

有参函数与无参函数的主要区别在于多了一个形式参数列表。在形式参数列表中给出的参数称为形式参数，我们简称它为形参，它们可以是各种类型的变量，之间用逗号分隔。在函数调用时，主调函数将传递实际的值给这些形式参数。

例如，定义一个函数，用于求两个数中的小数，可写为

```
int min(int a, int b)
{
  if (a<b) return a;
else return b;
```

```
}
```

程序第一行说明 min 函数的返回值为整型。a 和 b 为函数的形参。当 min 函数被调用时，上级函数会将实际的值传给形参 a 和形参 b。函数体中 return 语句的作用是，如果 a < b，就把 a 作为函数返回值，否则就把 b 作为函数返回值。

6.4.2 函数的参数和函数的值

形式参数和实际参数：

前面已经介绍过，函数的参数分为形参和实参两种。那么形参和实参的特点和两者的关系到底是怎样的呢。可以这样讲，形参只出现在函数定义中，在该函数体内可以进行有效地使用，在该函数之外，则不能使用。实参只出现在主调函数中，发生函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送，如图 6-1 所示。

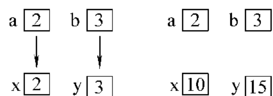


图 6-1 数值传递示意图

函数的形参和实参有以下特点：

1）形参只有在被调用时才临时分配内存单元，调用结束时，释放自己所占的内存单元。

2）实参必须有一个最终值，不管它是哪种类型的变量，在进行函数调用时，它都必须有确定的值，从而把这些值传送给形参变量。

3）实参和形参的数据类型必须一致，否则程序在编译时，将发生类型不匹配的错误。

4）函数调用的过程中只能将实参的值赋给形参，反过来是不可以的。

【例 6-8】

```
main( )
{
    int a;
    printf("input a number\n");
    scanf("% d",&a);
    m(a);
    printf("% d\n",a);
}
int m(int a)
{
    int i;
    a = a * a;
    printf("% d\n",a);
}
```

例 6-8 中定义了一个函数 m，该函数的功能是求 a 的平方值。在主函数中输入 a 值，并作为实参，在调用时传送给 m 函数的形参 a，在此说明一下，形参和实参变量名可以同样，

但它们是两个完全独立的变量。在函数 m 中，形参 a 的值已经被改写，即 a = a * a；同时，函数体中的 printf 语句将 a 值打印出来，当函数 m 返回主函数后，主函数中的 printf 语句又将 a 值打印出来，有可能你会想到这里的值和刚才函数 m 中打印值一样，但在 C 语言中，形参是不能改变实参变量的值，因此，主函数中的 printf 语句所打印的 a 值仍为我们最前面输入的值。

6.4.3 函数的返回值

函数的值是指函数被调用后，最终返回给主调函数的一个值。比如，我们调到的是一个取最小数的函数，那么函数返回的就是那个最小的值；如果调用的是一个开平方的函数，那么函数返回的是那个数开平方用的值。返回函数值时，请注意以下几点：

1）通过 return 语句返回函数值。

return 语句的一般形式为

```
return 表达式;
```

或者为

```
return（表达式）;
```

该语句的功能是将要返回的值回送给主调函数，当然返回值的类型必须和函数定义时的类型相一致。

2）如在函数定义时没有写明函数类型，则默认为整型。

3）如函数不需要返回值，函数类型可以写作“void”，表示该函数没有返回值。

如将例 6-8 中函数 m 定义为以下形式

```
void m(int a)
{ ......
}
```

6.4.4 函数的调用

1. 函数调用的一般形式

C 语言中，函数调用的一般形式为

```
函数名（实际参数表）
```

如果是无参函数，那就不存在以上的“实际参数表”。如果是有参函数，那么实际参数表中的参数可以是常数，变量或其他构造类型数据及表达式。各参数之间用逗号分开。

2. 函数调用的方式

函数定义好以后，要被其他函数调用了才能被执行。C 语言的函数是可以相互调用的，但在调用函数前，必须对函数的类型进行说明，就算是标准库函数也不例外。标准库函数的说明按功能不同分别写在不同的头文件中，使用时在文件最前面用#include 预处理语句引入相应的头文件。如前面一直使用的 printf 函数说明就是放在文件名为 stdio. h 的头文件中。调用就是指一个函数体中引用另一个已定义的函数来实现所需要的功能，这时函数体称为主调用函数，函数体中所引用的函数称为被调用函数。一个函数体中可以调用数个其他的函数，这些被调用的函数同样也可以调用其他函数，也可以嵌套调用。在 C51 语言中有一个函数

是不能被其他函数所调用的，它就是 main 主函数。

在 C 语言中，可以用以下几种方式调用函数：

1）函数表达式：例如：c = min（a，b）是一个赋值表达式，把函数 min 的返回值赋予变量 c。

2）函数语句：例如：printf（“%d”，a）；即直接写上函数名加上分号。

3）函数实参：例如：printf（“%d”，min（x，y））；即将函数作为另一个函数调用时的实际参数。该语句的作用是将 min 函数的返回值作为 printf 函数的实参。

6.4.5 被调用函数的声明和函数原型

在主调函数中调用某函数之前应对该被调函数进行声明，类似于使用变量之前要先进行变量的定义申明。这样做的目的是为了使编译系统能知道被调函数返回值的类型，以便在主调函数中对返回值做相应的处理。

其一般形式为

类型说明符 被调函数名（类型 形参，类型 形参...）；

或为

类型说明符 被调函数名（类型，类型...）；

括号内给出了形参的类型和形参名，或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。

如在 main 函数中对 min 函数的说明为

```
int min（int a，int b）;
```

或者

```
int min（int，int）;
```

C 语言规定在以下几种情况时可以省去主调函数中对被调函数的函数说明。

1）如果被调函数的返回值类型是整型或字符型时，可以不作说明，而直接调用它。C 编译器将自动对被调函数返回值按整型处理。

2）程序中，被调函数的定义的位置出现在主调函数之前时，这时也可以不对被调函数做说明而直接调用。

3）如在所有函数定义之前，在主函数外先说明了各个函数的类型，那么在后面的各主调函数中，可以不再对被调函数做说明。例如：

```
char x(int a);
float y(float b);
main()
{
……
}
char x(int a)
{
......
```

```
}
float u(float b)
{
……
}
```

因为 x 函数和 y 函数已在程序代码段的首行进行了预先说明。因此在后面各函数中无须对 x 和 y 函数再做说明。

4）对系统库函数的调用不需再加说明，但必须把该函数相应的头文件用 include 命令包含于源文件前部。

6.4.6 函数的嵌套调用

函数的嵌套调用是指在执行被调用函数时，被调用函数又调用了其他函数。这与其他语言的子程序嵌套调用的情形是类似的。其关系如图 6-2 所示。

图 6-2 表示了双层嵌套的程序执行情况。其执行过程是：main 函数中调用 a 函数时，在执行 a 函数过程中，a 函数调用了 b 函数，b 函数执行完毕后再返回 a 函数的转出点继续执行剩余代码，a 函数执行完毕后再返回 main 函数的转出点继续执行剩余代码。

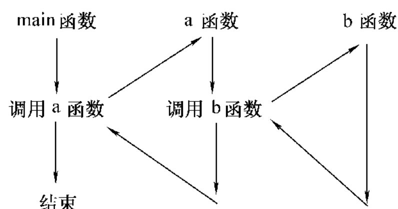


图 6-2 函数的嵌套调用

【例 6-9】 函数的嵌套调用。

```
main()
{
p1();/*在主函数中调用 p1()函数*/
}
p1()
{
p2();/*在 p1()函数中调用 p2()函数*/
printf("欢迎使用本程序! \n");
p2();/*调用 p2()函数*/
}
p2()
{
printf("*******************\n");
}
```

例 6-9 中，函数的调用关系是：main 函数调用 p1（）函数，p1（）函数调用 p2 函数。p2（）函数执行完之后，即返回到调用它的 p1（）函数中，继续执行调用处后面的语句。同样，p1（）函数执行完后，返回到调用它的 main（）函数中。

6.4.7 函数的递归调用

函数的递归调用是指一个函数在它的函数体内，直接或间接地调用它自身。

C 语言允许函数的递归调用。在递归调用中，调用函数又是被调用函数，执行递归函数将反复调用其自身。每调用一次就进入新的一层。

为了防止递归调用无终止地进行，必须在函数内有终止递归调用的手段。常用的办法是加条件判断，满足某种条件后就不再作递归调用，然后逐层返回。

例如有函数 f 如下：

```
int a(int x)
{
   int y;
   z = a(y);
   return z;
}
```

这就是一个递归函数。但是这个函数将永远不断地调用自身，从而进入了死循环，这当然是我们所不希望的结果。为了防止递归调用出现死循环，必须在函数内有终止递归调用的语句。一般用 if 语句进行条件判断，满足某种条件后就不再作递归调用，然后逐层返回。下面举例说明递归调用的执行过程。

【例 6-10】 计算 xn（x 和 n 均为正整数）。

```
main()
{
int a,y;
scanf("%d,%d",&a,&y);
printf("%d**%d=%d\n",a,y,power(a,y));
}
power(int x,int n)
{
int p;
if(n>0)
p=power(x,n-1)*x;
else
p=1;
return(p);
}
```

例 6-10 中，power（）函数的执行过程中，通过 power（x，n-1）直接调用了它自己。注意，程序中递归调用的条件是 n>0，当这个条件不再满足时（n=0），即终止了递归调用。

递归调用的优点是使程序简洁、紧凑，但由于每次调用一个函数时都需要存储空间来保存调用“现场”，以便后面返回，并且递归调用往往涉及到同一个函数的反复调用，所以它要占用很大的存储空间，特别是在递归调用次数较多的情况下，将导致程序运行速度较慢。

6.4.8 数组作为函数参数

前面已经讨论过如何用变量作为函数的参数，其实数组可以作为函数的参数使用，而进行数据传送。用数组作函数参数有两种形式，一种是把数组元素作为实参使用；另一种是把数组名作为函数的形参或实参使用。

1. 数组元素作函数实参

它与普通变量一样，没什么区别，因此它作为函数实参时的使用方法与普通变量是相同的，在发生函数调用时，把作为实参的数组元素的值传送给形参，实现单向的值传送。

【例 6-11】 写一函数，统计字符串中字母的个数。

```
/* 功能：数组元素作为函数实参 */
int isalp(char c)
{ if  (c>='a'&&c<='z'||c>='A'&&c<='Z')
          return(1);
  else     return(0);
}
main()
{ int i,num=0;
      char str[255];
      printf("Input a string: ");
      gets(str);
      for(i=0;str[i]!='\0';i++)
      if (isalp(str[i]))     num++;
      puts(str);
      printf("num=%d\n",num);
      getch();
}
```

例 6-11 中首先定义一个整型函数 isalp（），并说明其形参 c 为字符型变量。在函数体中根据 if 语句判断输出相应的结果。在 main 函数中用一个 gets 语句输入字符给 str，然后通过循环语句调用 isalp 函数，将其返回值用来统计字符串口字母的个数。

说明：

1）用数组元素作实参时，只要数组类型和函数的形参类型一致即可，并不要求函数的形参也是下标变量。换句话说，对数组元素的处理是按普通变量对待的。

2）在普通变量或下标变量作函数参数时，形参变量和实参变量是由编译器分配的两个不同的内存单元。在函数调用时发生的值传送，是把实参变量的值赋予形参变量。

2. 数组名作为函数参数

用数组名作函数参数与用数组元素作实参有以下不同：

1）用数组元素作函数参数时的处理是按普通变量对待的，但用数组名作函数参数时，则要求形参和相对应的实参都必须是类型相同的数组，都必须有明确的数组说明。当形参和实参二者不一致时，即会发生错误。

2）在普通变量或数组元素作为函数参数时，变量是由编译器分配的两个不同的内存单元。在函数调用时，实参变量的值将赋予形参变量。在用数组名作为函数参数时，不是进行值的传送，并不是把实参数组的每一个元素的值都赋予形参数组的各个元素。因为实际上形参数组并不存在，编译器不会为形参数组分配内存。看到这里，或许你会问，那么数据到底是如何传送的呢？我们曾介绍过，数组名就是数组的首地址。数组名作函数参数时，其实是将地址进行了传送，也就是说把实参数组的首地址赋予形参数组名。因此，形参数组和实参数组其实就是同一个数组，它们共同占用一段内存空间。两者之间的关系如图 6-3 所示。

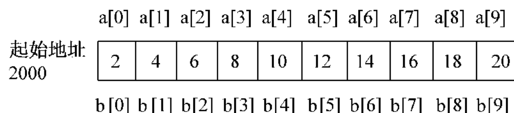


图 6-3　数组元素分布

图 6-3 直观地显示了它们之间的关系。图中设 a 为整型实参数组。数组 a 的起始地址为 2000，由此开始连续若干内存空间被 a 所占用。b 为形参数组。当函数被调用时，进行地址传递，把实参数组 a 的首地址传送给形参数组名 b，于是 b 的首地址便为 2000。于是 a，b 两数组共占 2000 为首地址的一段连续内存单元。从图中还可以看出 a 和 b 下标相同的元素实际上也占相同的两个内存单元（整型数组每个元素占 2 个字节）。例如 a［1］和 b［1］都占用 2002 和 2003 单元，当然 a［1］等于 b［1］。类推则有 a［i］等于 b［i］。

【例 6-12】　有一个一维数组 score，内放 10 个学生成绩，求平均成绩。

```
float average(float array[10])
{ int i;
  float aver,sum = array[0];
  for(i = 1;i < 10;i + + )
  sum = sum + array[i];
  aver = sum/10;
  return(aver);
}
void main(void)
{ float score[0],aver;
  int i;
  printf("input 10 scores:\n");
  for(i = 0;i < 10;i + + )
  scanf("%f",&score[i]);
```

```
printf(" \n");
aver = average(score);
printf("average score is %5.2f",aver);
}
```

运行情况如下：

```
input 10 scores:
56 78 98.5 76 87 99 67.5 75 97↙

average score is 83.40
```

例 6-12 首先定义了一个实型函数 score，有一个形参为实型数组 array，长度为 10。在函数 average 中，把各元素值相加求出平均值，返回给主函数。主函数 main 中首先完成数组 score 的输入，然后以 score 作为实参调用 average 函数，函数返回值送 aver，最后输出 aver 值。从运行情况可以看出，程序实现了所要求的功能。

3）前面我们已经讲过，在变量作函数参数时，传送的值是单向的，即只能从实参传向形参，而不能从形参传回实参。形参的初值和实参相同，如果程序中形参的值发生了改变，但实参值仍保持不变。而当用数组名作函数参数时就不一样了。由于形参和实参共享一组内存空间，因此当形参数组发生变化时，实参数组也随之变化。

6.5 局部变量和全局变量

我们在前面的学习中，使用了很多的变量。但是任何一个变量都是有它的管辖范围的（作用域），就是说，在定义了一个变量以后，并不是在程序的任何地方都可以使用这个变量。只有在变量的作用域内才能使用这个变量。在 C 语言中如果按作用域分，变量分为局部变量和全局变量。

6.5.1 局部变量

在一个函数内部定义的变量被称作局部变量，这种变量的作用域是在本函数范围内。通俗一点说，局部变量只能在定义它的函数内部使用，而不能在其他函数内使用这个变量。

```
float ff1(int a)
{
    int b,c;
}
```

在函数 ff1 内部变量 a,b,c 有效

```
float ff2(int x,int y)
{
```

```
        int x1,y1;
    }
```

在函数 ff2 内部 变量 x,y,x1,y1 有效

```
main()
    {
        int m,n;
    }
```

在函数 main 内部，变量 m，n 有效。

说明：

1）main 函数也是一个函数，它内部定义的变量也只能在 main 函数内部使用，而不能在其他函数内部使用 main 函数内部定义的变量。

2）不同的函数中可以使用相同的变量名，但它们是不同的变量。记得函数在执行时，系统要给它分配一块单独的内存吗？所以虽然变量名是相同的，但系统看到它们定义在不同的函数中，就认为它们是不同的变量。这样做有个好处，可以在函数内部，根据需要设置任何的变量名。如果不是这样，那么在一个函数内定义了一个变量名后，在其他函数内就不能再使用相同的变量名了。

3）形参也属于局部变量，作用范围在定义它的函数内。所以在定义形参和函数体内的变量时是肯定不能重名的了。

4）在复合语句内部也可以定义变量，这些变量的作用域只在本复合语句内。只在需要的时候再定义变量，这样做可以提高内存的利用率。

```
f(int a)
    {
{int c;c=a+b;}/*变量 c 只在此复合语句内起作用*/
    }
```

【例 6-13】

```
main ()
{
    int i=3, j=3, k;
    k=i+j;
    {
      int k=9;
      printf ("%d\n", k);
    }
    printf ("%d\n", k);
}
```

例 6-13 在 main 中定义了 i，j，k 3 个变量，其中 k 未赋初值。而在复合语句内又定义了 1 个变量 k，并赋初值为 9。注意这两个 k 是两个变量。在复合语句外程序部分，由 main 函数中定义的 k 起作用，而在复合语句内则由在复合语句内定义的 k 起作用。因此，程序第 4 行的 k 为 main 所定义，其值应为 6。第 7 行输出 k 值，因为它在复合语句内，由复合语句内定义的 k 起作用，其初值为 9，所以输出值为 9。第 9 行输出 k 值，因为在复合语句之外，输出的 k 应为 main 所定义的 k，k 值由第 4 行已获得为 6，故输出也为 6。

6.5.2 全局变量

全局变量也称为外部变量，它是在函数外部定义的变量。它不属于哪一个函数，它属于一个源程序文件，全局变量可以为本文件中其他函数所共用，它的有效范围为从定义变量的位置开始到本源文件结束。

例如：

```
int aa,bb;                /*外部变量*/
void f1()                 /*函数 f1*/
{
......
}
float xx,yy;              /*外部变量*/
int f2()                  /*函数 fz*/
{
......
}
main()                    /*主函数*/
{
......
}
```

从上例可以看出 aa、bb、xx、yy 都是在函数外部定义的外部变量，是全局变量。但大家注意到 xx，yy 定义在函数 f1 的后面，f1 函数内没有对 xx，yy 的说明语句，所以 xx，yy 在 f1 内是无法使用的。aa，bb 定义在源程序首行，因此在 f1、f2、main 函数中无需加入说明语句了。

【例 6-14】 外部变量与局部变量同名时的情况。

```
int a=3, b=5;             /*a，b 为外部变量*/
max (int a, int b)        /*a，b 为外部变量*/
{int c;
c=a>b? a: b;
return (c);
}
```

```
main ( )
{int a =8;
printf ( "%d\n", max (a, b));
}
```

运行结果：printf 打印出的字数为 8。

说明：

1）设全局变量的作用是为了增加函数间数据联系的渠道；

2）建议不在必要时不要使用全局变量，因为：

① 全局变量在程序的全部执行过程中都占用存储单元，而不是仅在需要时才开辟单元；

② 它使函数的通用性降低了；

③ 使用全局变量过多，会降低程序的清晰性，往往难以清楚地判断出每个瞬时各个外部变量的值。

3）如果在同一个源文件中，外部变量与局部变量同名，则在局部变量的作用范围内，外部变量被“屏蔽”，它不起作用。

第 7 章　指针、结构体与共用体

7.1　指针和地址

指针是 C 语言中一个极其重要的概念，也是 C 语言程序设计的难点。

1. 指针的定义

指针变量是用于存放其他变量的地址，简称为指针。指针变量所包含的是内存地址，而该地址便是另一个变量在内存中存储的位置。指针本身也是一种变量，和其他变量一样，要占有一定数量的存储空间，用来存放指针值（即地址）。

2. 指针的作用

C 语言中引入指针类型变量，不仅简单地实现了类似于机器间址操作的指令，更重要的是使用指针可以实现程序设计中利用其他数据类型很难实现，甚至无法实现的工作。指针的作用主要体现如下：

1）使代码更为紧凑和有效，提高了程序的效率。

2）便于函数修改其调用参数，为函数间各类数据传递提供简捷而便利的方法。

3）实现动态分配存储空间。

正确地使用指针会带来许多好处，但指针操作也有难以掌握的一面，若使用不当，可能会产生程序失控甚至造成系统崩溃。

7.2　指针变量和指针运算符

1. 指针变量及其说明

指针变量是存放地址的变量。和其他变量一样，必须在使用之前，加以定义说明。其定义说明的一般形式为

类型说明符 *指针变量名；

例如：

int *P；说明指针变量 P 是指向 int 类型变量的指针。

char *s；说明指针变量 s 是指向 char 类型变量的指针。

float *f；说明指针变量 f 是指向 float 类型变量的指针。

Double *d；说明指针变量 d 是指向 double 类型变量的指针。

static int pa；说明指针变量 pa 是指向 int 类型变量的指针，而 pa 本身的存储类型是静态变量。

int *fpi（）；说明 fpi（）是一个函数，该函数返回指向 int 类型变量的指针。

Int（*pfi）（）；说明 pfi 是指向一个函数的指针，该函数返回 int 类型变量。

指针变量值表达的是某个数据对象的地址，只允许取正的整数值。然而，不能因此将它与整数类型变量相混淆。所有合法指针变量应当是非 0 值，如果某指针变量取 0 值，即为

NULL，则表示该指针变量所指向的对象不存在。这是 NULL 在 C 语言中又一特殊用处。

2. 指针运算符 & 和 *

指针变量虽是正的整数值，但不是整数类型变量。指针变量最基本的运算符是 & 和 * 。

1）&——取地址。它的作用是返回后随变量（操作数）的内存地址。请注意，它只能用于一个具体的变量或数组元素，不可用于表达式。

例如：

```
int *p;
int m;
m=200;
p=&m;
```

这是将整数类型变量 m 的地址赋给整数类型的指针变量 p

2）*——取值。它的作用是返回后随地址（指针变量所表达的地址值）中的变量值。

例：

```
int *p;
int m;
int n;
m=200;
p=&m;
n=*p;
```

这是将指针变量 p 所指向的整数类型变量（即 m）的值赋给整数类型变量 n。其结果与赋值语句 m=n；一样，但操作过程用了指针变量 p。

& 与 * 运算符互为逆运算。对指针变量 px 指向的目标变量 * px 实行取址运算：

&（*px）

其结果就是 px。对变量 x 的地址实行取值运算；

*（&x）

其结果就是 x。

注意：x 与 * px 同级别可写成 x = * px 或 * px = x. px，与 &x 同级别可写成 px = &x，但决不可写成 &x = px，也不能写成 px = x。

3. 指针的初始化和赋值运算

指针的初始化往往与指针的定义说明同时完成，初始化一般格式为

类型说明符　指针变量名 = 初始地址值；

例如：

```
char c;
char *charp=&c;
```

请注意：

1）任何一个指针在使用之前必须加以定义说明；必须经过初始化；未经过初始化的指针变量禁止使用。

2）在说明语句中初始化，也是把初始地址值赋给指针变量，只有在说明语句中，才允许这样写。而在赋值语句中，变量的地址只能赋给指针变量本身。

3）指针变量初始化时，变量应当在前面说明过，才可使用 & 变量名作为指针变量的初始值。否则，编译时将出错。

4）必须以同类型数据的地址来进行指针初始化。指针变量可以进行赋值运算，这种赋值运算操作也仅限制在同类之间才可实现。

【例 7-1】 指针变量的引用。

```
main( )
{
int x =10
int  * p1 =&x, * p2; / *  p1 指向 x  * /
p2 =p1; / *  p2 指向 x  * /
printf("% d\n", * p2);
}
```

执行结果应显示 10

【例 7-2】 指针变量的运算。

```
main( )
{
char c ='A'; / * 变量 c 的初始值为'A' * /
char  * charp =&c; / * 变量 c 的地址作为指针变量 charp 的初始值 * / printf("% c% c\n",c,
 * charp);
c ='B'; / * 变量 c 赋值为'B' * /
printf("% c% c\n",c, * charp);
 * charp ='a'; / * 将指针变量 charp 所指向地址的内容改为'a' * / printf("% c% c\n",c, *
charp);
}
```

执行结果应显示

AA

BB

aa

4. sizeof 运算符

sizeof 运算符可以准确地得到在当前所使用的系统中某一数据类型所占字节数。sizeof 运算的格式为

sizeof（类型说明符）

其值为该类型变量所占字节数。用输出语句可方便地打印出有关信息，例如：

```
printf（ "% d\ n", sizeof（int））;
printf（ "% d\ n", sizeof（float））;
printf（ "% d\ n", sizeof（double））;
```

5. 指针的算术运算

指针的算术运算是按地址计算规则进行。由于地址计算与相应数据类型所占字节数有关，所以，指针的算术运算应考虑到指针所指向的数据类型。指针的算术运算只有如下 4 种：

+，-，++，--

1）指针自增 1 运算 + + 和指针自减 1 运算 - -：指针自增运算意味着指针向后移动一个数据的位置，指向的地址为原来地址 + sizeof（类型说明符）。例如：

```
int i, * p;
p = &i;
p + +;
```

结果是 p 指向了变量 i 的地址 + sizeof（int）。

2）指针变量的 + 和 - 运算：指针变量 px 加（+）正整数 n，表示指针向后移过 n 个数据类型，使该指针所指向的地址变为原指向的地址 + sizeof（类型说明符）* n。指针变量减（-）正整数 n，表示指针往前移回 n 个数据，使该指针所指向的地址变为原指向的地址 - sizeof（类型说明符）* n。

6. 指针的关系运算

指向同一种数据类型的指针，才有可能进行各种关系运算。指针的关系运算符包括有：

= =，! =，<，，> =.

它们实际所进行的是两个指针所指向的地址的比较。不同类型指针之间的比较是没有意义的。指针与整数常量或变量的比较也没有意义。唯独常量 0 是例外的，一个指针变量为 0（NULL）表示该指针指向空间不存在，称为空指针。如果有一个指针 p，可以用 p = =0 或 p！ =0 来判断 p 是否为空指针。

7. 使用指针编程中的常见错误

1）使用未初始化的指针变量，例如：

```
main( )
{
int x, * p;
x =0;
* p = x;      /* 指针变量未初始化 */
  …
}
```

* p 未被初始化。

2）指针变量所指向的数据类型与其定义的类型不符，例如：

```
main( )
{
float x,y;
short int  * p;
p = &x;/ *   / * 数据类型不符 * /
y = * p;
  …
```

```
}
```

x,y 与 * p 数据类型不符。

3）指针的错误赋值，例如：

```
main( )
{
int x, * p;
x = 10
p = x
  …
}
```

错在以普通变量 x 赋给指针变量 p。

4）用局部变量的地址去初始化静态的指针变量，例如：

```
int glo;
func( )
{
int 1oc1,1oc2;
static int  * gp = &glo;
static int  * 1p = &1oc1;
int  * rp = &1oc2;
  …
}
```

用局部变量 1oc1 去初始化静态的指针变量 1p 是错误的。

7.3 指针与函数参数

在函数调用中，使用指针作为参数，可以改变调用环境中的变量之值。这是使用一般变量作为参数所难以实现的。让我们来观察一个函数的调用：

【例 7-3】 x 和 y 的值互换。

```
swap1(x,y)
int x,y;      /* 局部变量不能带出函数 */
{
int temp;
temp = x;
x = y;
y = temp;
}
main( )
{
```

```
int a,b;
a=10;
b=20;
swap1(a,b);
printf("a=d% d b=% d\n",a,b);
}
```

执行结果：

a=10 b=20

虽然，swap1（）函数中，x 和 y 的值互相交换了，但 x，y 是局部变量，主函数对该函数的调用 swap1（a，b）是“传值”调用，x 与 y 的交换结果，不可能影响调用者中的变量 a，b，a 与 b 之值并未交换。

改用指针作为参数，也就是用变量的地址作为参数，即所谓“传址”（或称为“传名”）的调用，就可改变相应地址中的变量。

【例 7-4】 x 和 y 的值互换。

```
swap2(px,py)
int *px,*py;  /* 地址变量 */
{
int temp;
temp= *px;
*px= *py;
*py=temp;  /* 返回后地址变量释放*/
}
main()
{
int a,b;
a=10;
b=20;
swap2(&a,&b);  /*传地址*/
printf("a=%d b=%d\n",a,b);  /*得到交换的值*/
}
```

执行结果：

a=20 b=10

a 与 b 的地址，即 &a 与 &b 并未交换，但其值已被交换。

7.4 指针、数组和字符串指针

1. 指针和数组

用指针和用数组名访问内存的方式几乎完全一样，却又有微妙而重要的差别。指针是地

址变量，数组名则是固定的某个地址，是常量。

若定义一个数组：

int a［10］；

则在内存中占有从一个基地址开始的一块足够大的连续的空间，以包容 a［0］，a［1］…，a［9］。基地址（或称为首地址）是 a［0］存放的地址。其他依下标顺序排列。

若定义一个指针：

int ＊p；

则意味着：p＝&a[0]；即

p 指向 a[0]

p＋1 指向 a[1]

p＋2 指向 a[2]

…

p＋i＝&a[i]；

p＋i 指向 a[i]（i＝0，…，9）

1）在 C 语言中，数组名本身是指向零号元素的地址常量。因此，p＝&a［0］可写成 p＝a，a 当作常量指针，它指向 a［0］. p＝a＋i 与 p＝&a［i］等价，故数组名可当指针用，它指向 0 号元素。

2）若定义 p 为指针，a 为数组名，且 p＝a，则 p＋i 指向 a［i］，a＋i 也指向 a［i］。这样，a［i］与＊（p＋i）或＊（a＋i）可看成是等价的，可互相替代使用。

注意：

由于 a 只是数组名，是地址常量，而不是指针变量，所以可以写 p＝a，但不能写 a＝p，可以用 p＋＋，但不能用 a＋＋。

2. 字符串指针

在 C 语言中，可用 char 型数组来处理字符串，而数组又可用相应数据类型的指针来处理。所以可以用 char 型指针来处理字符串。通常把 char 型指针称为字符串指针或字符指针。

【例 7-5】 计算串长度的函数 strlen（）是使用字符串指针的一个实例。

```
strlen(s)
char *s;
{
int n;
for(n=0;*s! ='\0';s++)
n++
return(n);
}
main()
{
static char str[]={"Hello!!"};
printf("%c\n",*str);
```

```
printf("%d\n",strlen(str));
printf("%c\n",*(str+1));
}
```

执行结果：

H

6

e

定义字符指针可以直接用字符串作为初始值，来初始化字符指针。例如：

```
char *message="Hello!";
```

或

```
char *message;
message="Hello!";
```

这样的赋值并不是将字符串复制到指针中，只是使字符指针指向字符串的首地址。但对于数组操作，例如：

```
char text[80];
```

则不可写成：

```
text="Hello!";
```

因为，此处 text 是数组名，而不是指针，只能按字符数组初始化操作。即：

```
static char text[80]={"Hello!"};
```

当字符串常量作为参数（实参）传递给函数时，实际传递的是指向该字符串的指针，并未进行字符串复制。

【例 7-6】 向字符指针赋字符串。

```
main()
{
char s="Hello!";/*相当于 s 数组*/
char *p;
while(*s!='\0')
printf("%c",*s++);
printf("\n");
p="Good-bye!";  /*指针指向字符串*/
while(*p!='\o')
printf("%c",*p++);
printf("\n")
}
```

执行结果：

Hello!

Good-bye!

此例中字符串是逐个字符输出,也可以字符指针为参量,作字符输出,例如：

```
printf("%s",s);
```

或

```
printf("%s",p);
```

输入函数 scanf( )也可以字符指针为参量,如:

```
scanf("%s",p);
```

【例 7-7】 以字符指针为参数来调用串比较函数 strcomp ( )。

```
strcomp(s,t)
char *s, *t;   /*形式参数为字符指针*/
{
for( ; *s = = *t;s + +,t + +)
if( *s = =‘\o’)
return(0);
return( *s- *t);  /* 字符串相同返回零值 */
}
main( )
{
static char s1[ ] ={"Hello!"};
char *s2 ="Hello!";
printf("%d\n",strcomp(s1,s2));
printf("%d\n",strcomp("Hello",s2));
}
```

s 和 t 初值已由实参传递过来了, 所以 for 语句的第一个表达式为空语句。

用指针处理字符串的复制, 比用数组处理更精练, 例如:

```
strcpy(s,t)
char *s, *t;
{
while( *s + + = *t + +);
}
```

如果用数组处理则要复杂些,例如:

```
strcpy(s,t)
char s[ ],t[ ];
{
int i;
i =0;
while((s[i] =t[i])!  =‘\o’)
i + +;
}
```

直接演化出指针处理,即:

```
strcpy(s,t)
char *s, *t;
{
while(( *s = *t)!  =‘\o’)
{
s++;
t++;
}
}
```

优化后便是如前所示的指针处理方法的程序。其中，也使用了空语句（;），而且不必进行与‘\0’的比较。

7.5 指针数组

1. 指针数组的定义和说明

同类指针变量的集合，就形成了指针数组。或者说，以指针变量为元素的数组，就称为指针数组。这些指针变量应具有相同的存储类型，并且，指向的目标数据类型也应相同。

指针数组的一般表示格式为

类型说明符 *指针数组名［元素个数］;

例如：

```
int *p[2];
```

p[2]是含有 p[0]和 p[1]两个指针的指针数组,指向 int 型数据。

2. 指针数组的初始化

指针数组的初始化可以在说明的同时进行。与一般数组一样，只有全局的或静态的指针数组才可进行初始化。而且，不能用局部变量的地址去初始化静态指针。

【例 7-8】 指针数组。

```
#include
main()
{
static int b[2][3]={{1,2,3},{4,5,6}};
static int *pb[]={b[0],b[1]};  /* 指针数组指向行首 */
int i,j;
for(i=0;i<2;i++){
for(j=0;j<3;j++)
printf("b[%d][%d]=%d",i,j,*(pb[i]+j));
printf("\n")
}
}
```

执行结果：
b[0][0]=1 b[0][1]=2 b[0][2]=3
b[1][0]=4 b[1][1]=5 b[1][2]=6
此例中，将一个二维数组 b[2][3]分解成两个一维数组。它们的首地址，分别为 b[0]和 b[1]，并被赋给指针 pb[0]和 pb[1]。

3. 字符指针数组

字符指针可以用来处理一个字符串。字符指针数组可以用来处理多个字符串。这正是字符指针数组的主要作用。

【例 7-9】

```
main( )
{void sort( );
void print( );
static char *name[ ] = {"Follow me","BASIC","Great Wall",
"FORTRAN","Computer design"}; /*指针数组指向各字符串*/
int n=5;
sort(name,n);
print(name,n);
}
void sort(name,n)
char *name[ ]; int n;
{char *temp;
int i,j,k;
for (i=0;i {k=i;
for (j=i+1;j0) /*指向各字符串的比较*/
k=j;
if (k! =i)
{temp=name[i]; name[i]=name[k]; name[k]=temp;}
}
}
void print(name,n)
char *name[ ]; int n;
{int i;
for (i=0;i printf("%s\n",name[i]); /*按排好序的指向输出字符串*/
}
```

运行结果：
BASIC
Computer design
FORTRAN

Follow me
Great Wall

7.6 多级指针

指针的指针，就形成了多级指针，也就出现了多级间址访问的现象。多级指针也称为指针链。一般很少使用超过二级的指针。过多的间址难于调试跟踪，也容易出错。常用的多级指针也只是二级指针，其定义说明的一般格式如下

类型说明符　* *指针变量名

例如：

int * *p;

这里 p 为二级指针,它所指向的是一个指向整数型数据的指针。又如：

static char * *charp;

说明 charp 本身是静态的二级指针,指向的是个指向最终目标为字符型变量的指针。

【例 7-10】　二级指针

```
main( )
{
int x, * p, * * q;
x =10;
p =&x;
q =&p;
printf("% d\n", *  * q);
}
```

用指针来处理数组是很自然的，在实际使用中，二级指针常用来处理字符指针数组。

【例 7-11】

```
#include
main( )
{
char * * pp;
static char * di[ ] = {"up","down","left","right",""}; /* 字符指针
数组指向字符串 */
pp = di; /* 二级指针 pp 指向字符指针数组 di */
while( * * pp!  = NULL)
printf("% s\n", * pp + + );
}
```

执行结果：

up

```
down
left
right
```

【例 7-12】 指针数组指向整型数组

```
main( )
{static a[5] = {1,3,5,7,9}; /* 整型数组 */
static int *num[5] = {&a[0],&a[1],&a[2],&a[3],&a[4]}; /* 指针数组刺 */
int **p ,i;
p = num; /* 指向指针数组 */
for(i = 0;i < 5;i + +)
{printf("% d\t", **p);p + +;}
}
```

运行结果:

```
1 3 5 7 9
```

7.7 返回指针的函数

一个函数被调用后，可以返回各种类型的数据，也可以返回指针数据，也就是地址。

【例 7-13】 有若干学生，每个学生 4 门课，要求输入学生序号后，能输出该学生的全部成绩。

```
main( )
{static float score[ ][4] = {{60,70,80,90,},{56,89,67,88},{34,78,90,66}};
float *search( ); /* 说明函数返回指针 */
float *p;
int i,m;
printf("enter the number of stuednt:");
scanf ("% d",&m);
printf("The scores of No. % d are:\n",m);
p = search(score,m); /* 得到该位同学的行首地址,指向列 */
for(i = 0;i < 4;i + +)
printf("%5.2f\t", *(p + i));
}
float *search(pointer,n)
float (*pointer)[4];
int n;
{float * pt;
pt = *(pointer + n); /* 指向列 */
```

```
return(pt);
}
```

运行结果：

```
enter the number of student:1↙
  The score of No. 1 are:
56.00 89.00 67.00 88.00
```

7.8 函数指针

1. 函数指针的定义和说明

函数也具有与数组类似的特性，可以用函数名表示函数的存储首地址，即执行该函数的入口地址。指向函数入口地址的指针，就称为函数指针。函数指针定义说明的一般格式如下

数据类型说明（＊函数指针名）()

例如：

```
int (*func)();
```

2. 函数指针的作用

对于指向某函数的函数指针 func，实现访问目标的运算＊，即（＊func）()，其结果是使程序控制转移到指针所指向的地址去执行该函数。所以，函数指针所指向的是程序代码区。而不是数据区。这与一般数据指针变量有原则的区别，一般数据指针指向数据区。

7.9 结构与联合

7.9.1 结构的定义

1. 结构的定义及其一般格式

数组是将同类型元素组成单个逻辑整体的一种数据类型。而结构则是将不同类型元素组成单个逻辑整体的一种数据类型。结构是 C 语言中一种强有力的数据类型，是很重要的概念之一。

例如，日期由年，月，日构成：

```
int year;
int month;
int day;
```

上述表示方法不能看出它们是彼此有关系的 3 个量。可以采用下面的定义表示

```
struct date
{
int month;
int day;
int year;
};
```

这样，就可把年，月，日当作一个整体处理。年，月，日是该结构的成员。结构的成员也可以是不同数据类型的变量，例如：

```
struct wage
{
char name[30];
float salary;
};
```

上面定义了两个结构数据类型，date 和 wage。可用这些已定义的结构类型来说明一组具体结构类型变量。例如：

```
struct date today,tomorraw;  /* 定义了两个结构类型变量 today,tomorraw */
struct wage worker1,worker2,worker3;/* 定义了 3 个结构类型变量 */
```

结构定义的一般格式为

```
struct[结构类型名]
{
类型说明符  成员变量名;
  …
类型说明符  成员变量名;
}[结构变量列表];
```

结构变量列表是指以逗号分隔开的若干结构类型变量，例如：

```
struct date
{
int month;
int day;
int year;
}today;
```

或

```
struct
{
int month;
int day;
int year;
}today;
```

或

```
struct date /*  定义了 1 种结构类型  */
{
int month;
int day;
int year;
```

```
};
struct date today;/*   说明了一个具体的结构变量   */
```

这 3 种说明结构类型变量的方式都可以使用，第 3 种写法的风格较好。

2. 结构的存取

结构变量成员可作为单独变量来操作，也就是说，可以直接访问结构中的一个成员变量，其格式为

结构变量名。成员变量名

【例 7-14】 显示输入的年，月，日。

```
#include
main( )
{
struct date /*   定义了结构类型   */
{
int month;
int day;
int year;
};
struct date today;/*   结构体变量 today   */
printf("Enter today's date(年,月,日)\n");
scanf("%d,%d,%d",&today.year,&today.month,&today.day);
printf("Today's date is %d/%d%/d\n",today.year, today.month,
today.day )
}
```

执行后可以显示出所输入的今天的年，月，日。

【例 7-15】 对外部存储类型的结构体变量的初始化。

```
struct student /*   定义结构类型   */
{ long int num;
char name[20];
char sex;
char addr[20];
} a = {89031,"Li Lin",'M',"123 Beijing Road"}; /*结构体变量赋初值*/
main ( )
{ printf("No. :%ld\nname:%s\nsex:%c\naddress:%s\n",a.num,a.name,
a.sex,a.addr);
}
```

【例 7-16】 对静态存储类型的结构体变量的初始化。

```
Main ( )
{
static struct student /*  静态结构类型  */
{long int num;
char name[20];
char sex;
char addr[20];
} a = {89031,"Li Lin",'M',"123 Beijing Road"}; /*赋初值*/
printf("No. :%ld\nname:%s\nsex:%c\naddress:%s\n",a. num,a. name,
a. sex,a. addr);
}
```

7.9.2 结构数组

不仅结构变量成员可以是数组，而且，同一类结构变量的集合还可构成结构数组。例如：

```
struct wage
{
char name[30];
float salary;
};
static wage persons[100];
```

这就说明了 100 个结构变量所组成的数组。

【例 7-17】 输入后选人名，对后选人投票计数，统计输出每个人的得票结果。

```
Struct person /* 全局结构类型 */
{char name [20];
int count;
} leader [3] = { "Li", 0, "Zhang", 0, "Fun", 0}; /*初始化*/
main ( )
{int I, j;
char leader_name [20];
for (I=1; I< =10; I+ +)
{scanf ( "%s", leader_name); /*输入人名*/
for (j=0; j<3; j+ +)
if (strcmp (leader_name, leader [j] . name) = =0)
leader [j] . count+ +; /*累加票数*/
}
printf ( "\ n");
for (I=0; I<3; I+ +)
```

```
printf ( "%5s:%d \ n", leader [I] . name, leader [I] . count); /* 每人得票数*/
}
```

7.9.3 结构与函数

1. 向函数传递结构变量成员

结构变量成员可作为参数传递给函数。例如有一结构为

```
struct fred
{
char x;
int y;
float z;
char s[10];
}mike;
```

其中各个成员均可作为函数的参数,例如:

```
func0(mike. x); /*字符型参数*/
func1(mike. y); /*整数型参数*/
func2(mike. z); /*浮点型参数*/
func3(mike. s); /*字符型数组参数*/
func4(mike. s[2]); /*字符型参数*/
```

2. 向函数传递完整的结构

不仅结构元素可作为参数传递给函数，整个结构也可作为参数传递的函数。完整结构向函数传递的操作中，应注意:

1）整个结构按传值方式传递。和普通变量一样，与数组不同，在函数内所引起结构参数中某个值的变化，只影响函数调用时所产生的结构拷贝，不影响原来的结构。

2）结构分全局定义和局部定义。定义在所有函数外部的结构称为全局结构，定义在函数内部的结构称为局部结构。它们被引用的范围不同。

【例 7-18】 局部定义的结构。

```
main()
{
struct  /*局部定义*/
{
int a,b;
char ch;
}arg;
arg. a =1000;
fl(arg);
printf("a =%d\n",arg. a);
}
```

```
fl(parm)
struct   /*形参说明定义*/
{
int x,y;
char ch;
}parm;
{   /*函数体*/
printf("x1 = %d\n",parm.x);
parm.x = 20;
printf("x2 = %d\n",parm.x);
return;
}
```

执行结果：

x1 = 1000

x2 = 20

a = 1000

【例 7-19】 全局定义的结构。

```
struct st   /*全局定义*/
{
int a,b;
char ch;
};
main()
{
struct st arg;
arg.a = 1000;
f1(arg);
}
f1(parm)   /*定义函数据库*/
struct st parm;   /*形参说明*/
{
printf("%d\n",parm.a);
return;
}
```

其中，结构类型 st 是全局定义的，各函数都可引用。

7.9.4 结构的初始化

简单结构的初始化

结构说明时，行尾加上分号，随后在花括号中按结构定义的各成员顺序给以各自的初始值，并用逗号分隔之。例如：

```
struct date
{
int month;
int day;
int year;
}; /* 行尾加上分号 */
static struct date today = {11,19,1991};
```

注意：局部结构变量不能初始化。

7.9.5 联合

1. 联合的定义说明及其一般格式

在 C 语言中，不同数据类型的数据可以共用同一个存储区。这是又一种构造类型的数据类型，称为联合（union）。联合定义说明的一般格式如下

```
union [联合类型名]
{
类型说明符 变量名;
   …
类型说明符 变量名;
} [联合变量列表];
```

例如：

```
union mixed
{
char c;
float f;
short int i;
}x;
```

或

```
union
{
char c;
float f;
short int i;
}x;
```

或

```
union mixed
{
char c;
```

```
float f;
short int i;
};
union mixed x;
```

从形式上看，联合的定义说明与结构很相似，然而，在内存的配合和使用上，两者有本质的区别。上述例子中的联合变量 X 所占字节数应为 3 个成员变量 c，f 和 i 之中占用字节数的最大值，即变量 f 所占字节数，可占 4 字节。同一时刻，只能存有 3 个成员变量之一。联合的好处是可以节省空间。

2. 联合的存取

联合成员的引用，在表示方法上，类似于结构成员的引用。其一般形式如下

联合类型变量名．成员变量名

第8章　51单片机实验器材快速操作入门

8.1　增强型51单片机实验板操作入门

增强型51实验板是杭州晶控电子有限公司（www. hificat. com）综合多年应用经验开发出的多功能8051单片机平台，如图8-1所示。集成了常用的单片机外围硬件资源。系统附带有众多汇编/C语言实例程序，可以让您在最短的时间内，全面地了解并掌握单片机的编程技术。特别适合于单片机的初学者，可供大中专院校师生，单片机工程师和实验室选用。

1. 增强型51单片机实验板板载硬件资源

增强型51单片机实验板板载实验部分包含了常见的、经典单片机实验硬件单元，完全兼容教科书及网上的各类单片机教程，包含的试验单元和接口单元如下：

1）数码管：可以试验和仿真各种计数器、数字显示，以及用单片机做电子钟等仿真。比如计数器、秒表、电子钟等。

2）LED流水灯：可以显示P口的状态，以及试验和仿真各种LED实验。比如正、反流水灯，交通指示等。

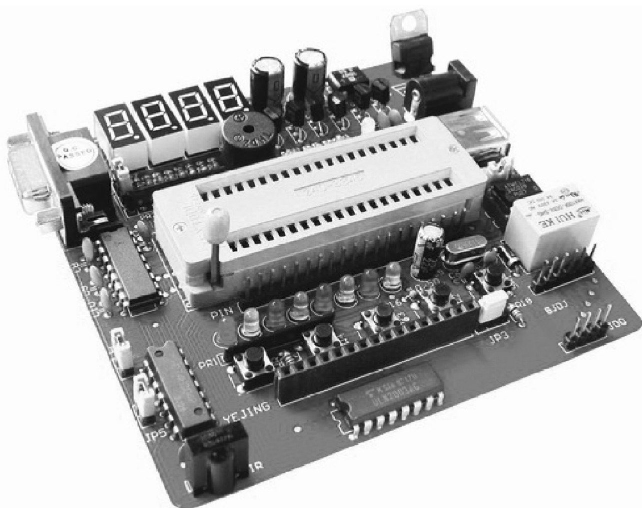


图8-1　增强型51单片机实验板

3）键盘：共4个键位，非常实用的键盘，通过简洁的程序即可完成键盘输入控制，编程方面更不需要像矩阵键盘那样绞尽脑汁。

4）蜂鸣器：音乐输出蜂鸣器可以完成各种奏乐、报警等发声音类实验。

5）继电器：我们可以通过继电器来做一个以弱控强的系统，以弱控强器件是工控中最常用器件之一，与其他驱动器件相比明显的优点是抗过载能力强，强弱端隔离能力强。

6）24C02：用来做IIC通信实验。

7）1602液晶屏接口：通过1602液晶屏显示你想要的信息，比发光管、数码管显示更为漂亮和专业化。

8）串口通信电路：单片机和计算机完成联机通信的接口。

9）板上含有步进电动机驱动电路，可以非常方便地接上步进电动机，完成步进电动机的各类实验，如电动机的正、反转等。

10）带红外接收头，可以做红外线解码实验，红外线遥控器等。配合遥控器完成遥控解码及红外遥控实验。如：按遥控器上的按键，即可点亮实验板上相应的发光管。当然，你

也可以通过改动程序来达到红外遥控其他资源的目的。

11）所有芯片引脚都接有外扩排针，有利于外扩更多的功能，外扩实验的功能没有限制，完全由用户决定。

12）支持 STC 单片机 ISP 在线下载功能，通过串口烧写芯片，非常方便。

13）红色 PCB，彩色跳线，做工精细，看上去非常漂亮。

14）支持 USB 转 RS232 串口：用 USB 口完成串口通信实验，可以直接用于只有 USB 口的笔记本电脑或台式计算机。

2. 系统软硬件需求

Windows98/ME/XP/2003 操作系统，最小硬盘空间为 80MB。

主要硬件接口功能说明：

RS232 串口：用于仿真操作（如无串口，可以用 USB 转 RS232 串口线）。

USB 口：提供实验板电源电压。

外接电源口：使用 9 ~ 12V 交流或直流外接电源。

3. 增强型 51 单片机实验板使用注意事项及跳线说明

1）在 USB 供电部分加了限流电阻。采用 USB 口供电时，最大电流限制到 500mA，有效地避免“过载和短路”等给电脑 USB 口造成的伤害。如果和仿真器联机的目标板的供电电流大于 500mA，请不要采用 USB 方式供电。

2）普通电源插座的后级接有全桥整流电路，输入的电源可以是交流或者直流，并且不分正、负极；输入电压范围是交流 8 ~ 12V，直流 8 ~ 15V。不论交、直流，保证在输出端可以得到稳定的 5V 标准电源电压。

增强型 51 单片机实验板上共有 5 个跳线，跳线的使用说明如下：

1）跳线 JP1：是电源类型选择跳线。位于右位时，表示通过 USB 口取得电源。位于左位时，表示从普通电源口取得电源。如果是选择由仿真器供电，请将此跳线置于右位，表示通过 USB 口取得电源。

2）跳线 JP2：是 P0 口上拉电阻选择跳线。位于下位时，表示连接 10kΩ 上拉电阻。位于上位时，表示断开 10kΩ 上拉电阻。

3）跳线 JP3：液晶屏背光灯跳线开关。位于左位时，液晶屏背光灯点亮，发出绿光。位于右位时，液晶屏背光灯不亮。

4）跳线 JP4、JP5：外接串口控制开关。两个都接跳线帽时，外接串口与实验板电路连通。两个拔掉跳线帽时，外接串口无效。

关于键盘，最左侧标号为 SWR 的按钮是复位按钮，其他的是连接 P 口的按钮。

4. 循序渐进的学习方法

当您拿到这套精美的 51 单片机学习板时，请不要急于通电，建议：

1）先用 1h 左右的时间仔细阅读相关的说明文档。

2）了解单片机学习器材的操作方法及各按钮、模块的使用说明。

3）使用配套光盘内所带的主机自检程序来测试一下系统的正常性。

4）从本套件的例子中挑选适合您了解程度的例子做实验，如果正常了，那么想想这个例子为什么要这样写？这句话不要可不可以？想好了再改程序，重新做实验。如果有问题，最好的办法是登陆 http：//www. hificat. com/bbs 访问论坛，随时有数 10 位热心朋友和你在

一起分享学习的喜悦和进步!

8.2 增强型51单片机实验板仿真操作指南

(由于Keil操作较复杂，如果您是新手，请仔细阅读以下说明书，一步步地操作，相信您一定能够成功!)

1. 概述

增强型51单片机实验板的仿真可以通过微型51仿真器来配合操作，与Keil软件联机使用。

1) 可仿真89C51、89C52、89S51、89S52、89C58等51内核的单片机。

2) 直接支持Keil C51的IDE开发仿真环境。

3) 监控程序占用用户的资源少，全速运行不占用资源。

4) 片内64KB程序空间可以随时进行在线程序更新。

5) 可单步、断点、全速、可参考变量、RAM变量。

6) 支持汇编、C语言，混合调试。

7) 板载仿真头接口可以和任何的试验板、目标板进行连接，从而达到硬件仿真的无限扩展。

2. Keil软件安装

安装仿真器软件——Keil，用户可以在配带的软件光盘“keil C51中文完全版”目录下找到，运行“Setup.exe”文件进行安装，无需特别的参数设置，按其默认值确认即可，具体安装方法可以看目录下的说明文件。安装完成之后，点击开始菜单“程序”中的“Keil uVision2”，进入软件界面，如图8-2所示。

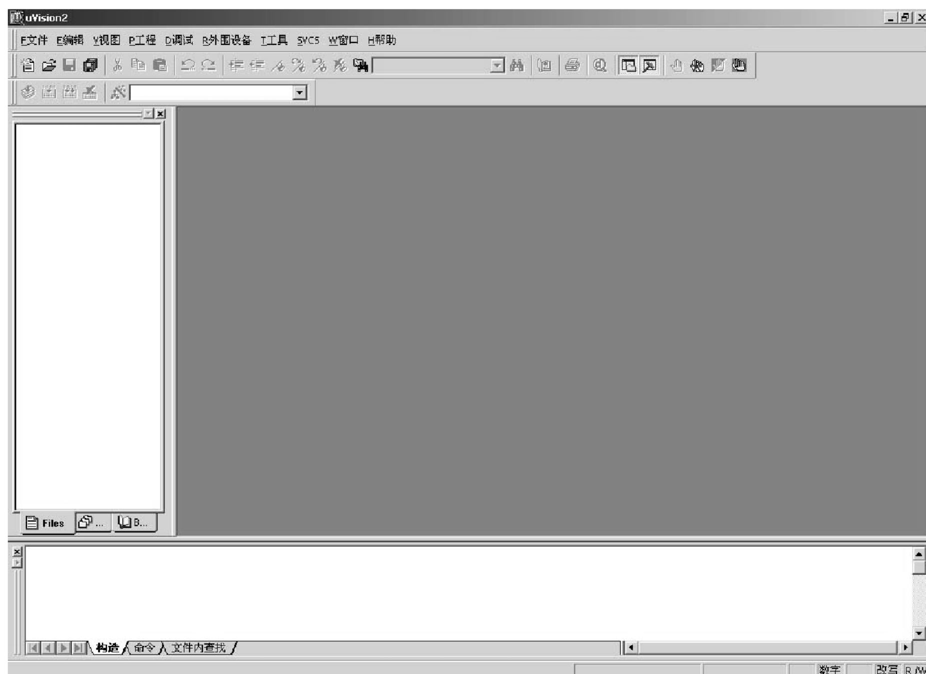


图8-2 Keil软件初始化界面

8.3 增强型51单片机实验板仿真实例

我们将与单片机的开发设备结合来讲述具体的实践和学习过程。

第一个实验是要用单片机点亮实验板上的第一只LED。用单片机来完成一些智能化的控制，这是一个最简单的程序实例，以给大家一个感性的认识。

实验板上共有8个LED，分别与单片机的P1.0脚～P1.7脚相连。现在我们就来点亮第一个发光管，即与P1.0脚相连的那个发光管。首先，我们将仿真器插上串口线，把串口线的另一头插至计算机的COM口上，并把仿真器插在51单片机实验板上，至此硬件设备连接完成。如图8-3所示。

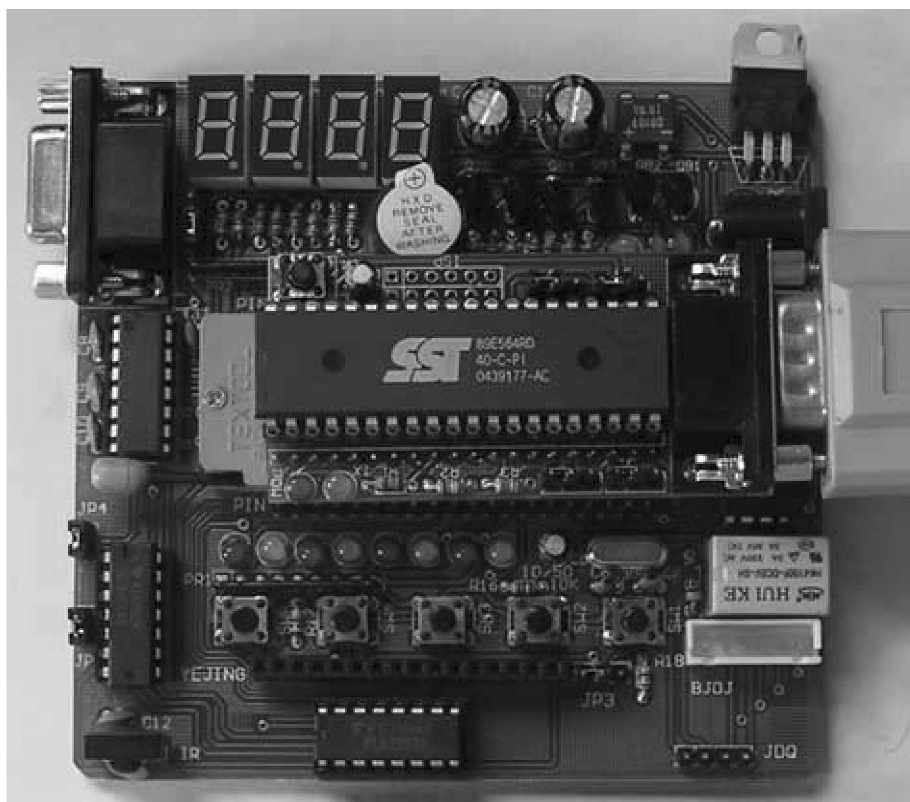


图8-3　单片机控制连线

接下来，我们安装仿真器软件——Keil，用户可以在配带的软件光盘“仿真软件Keil”目录下找到，运行“Setup.exe”文件进行安装，无需特别的参数设置，按其默认值确认即可，具体安装方法可看目录下的说明文件。安装完成之后，点击开始菜单“程序”中的“Keil uVision2”。进入软件界面，如图8-4所示。

我们在“工程”菜单中执行“新建”命令，新建工程文件名取为“my.uv2”。接下来选择我们要做实验使用的CPU类型，在此使用市面上最为常见的、Ateml公司的AT89C51型号，选好后点击“确定”按钮即可，这时工程向导已经做完。下一步将编写源程序代码，

即点亮第一个 LED 所需要的程序代码。我们执行“新建”操作，在弹出的文本编辑框内键入以下代码：

```
loop:
clr p1.0
ajmp loop
end
```

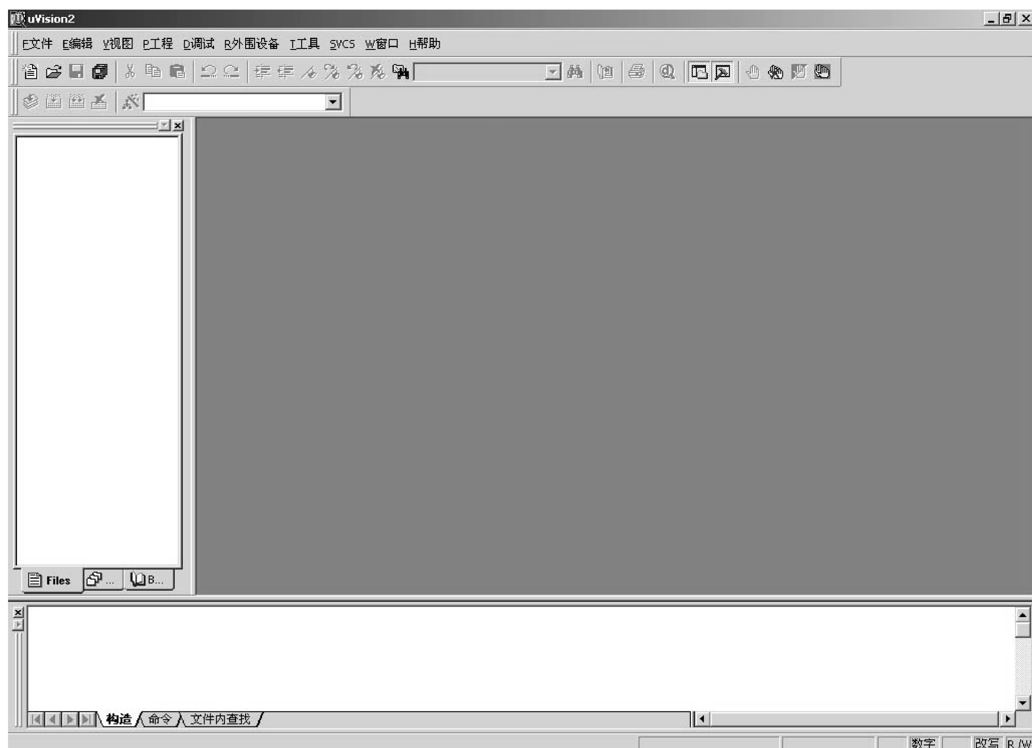


图 8-4　软件界面

这里我们仅使用了四条语句，CLR P1.0 的作用是使单片机的 P1.0 引脚置成低电平，因为要使第一个发光管点亮。从电路图中可以看到只要使 P1.0 脚为低电平信号即可。第一行的“loop”是语句标号；“ajmp loop”这条语句的意思是程序运行到此跳转到开始标号“loop”，重复执行程序。end 则是程序结束的标记。一个最简单的程序就这样编写完成了，然后将已经编好的程序保存，即执行“文件”菜单中的“另存为”命令，在此我们将文件名取为“led.asm”，注意 .asm 是汇编语言的扩展名，如果使用 C 语言编写的话，则扩展名应是“.c”。在此，我们先使用汇编语言来介绍，如图 8-5 所示。

“OK”，现在我们已经保存好了这个文件，还记得吗？我们刚才新建了一个叫“my”的工程，而“led.asm”文件应该是“my”这个工程的其中一分子，换句话说，我们还应该把这个“led.asm”文件添加到“my”这个工程当中去。具体操作如下：点击屏幕左侧的“Target1”字样旁边的“+”图标，则会弹出一个子项，名为“Source Group 1”，在其上面单击鼠标右键，选择“增加文件到组 Source Group 1”这项，将我们刚才保存的“led.asm”文件加进去，如图 8-6 所示。

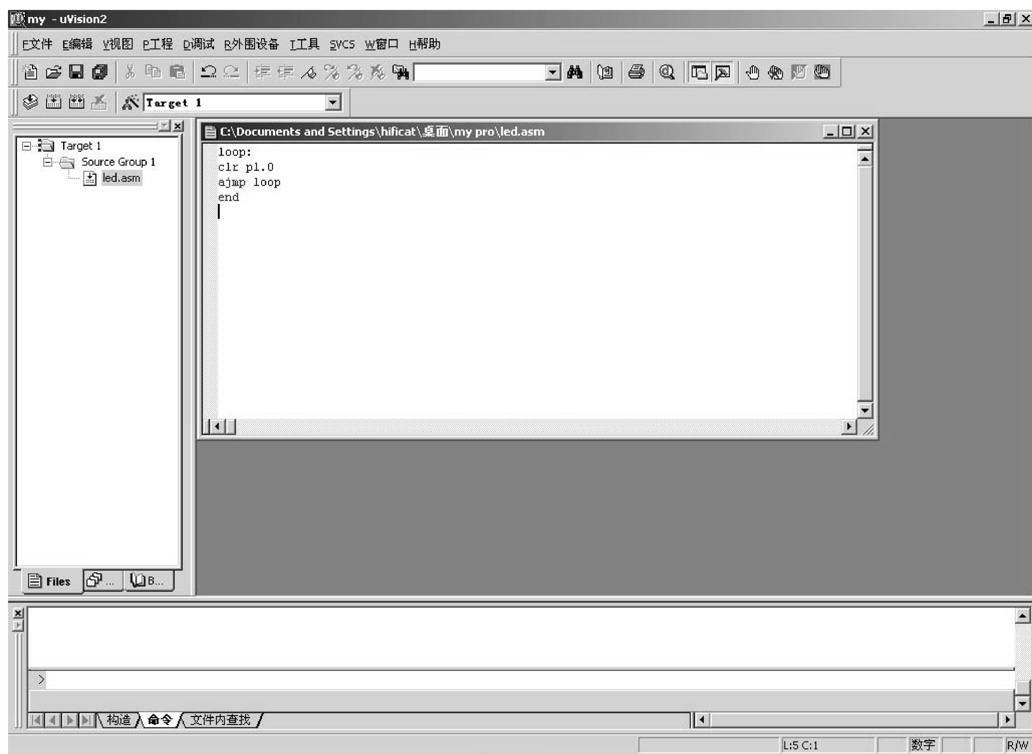


图 8-5　编程界面

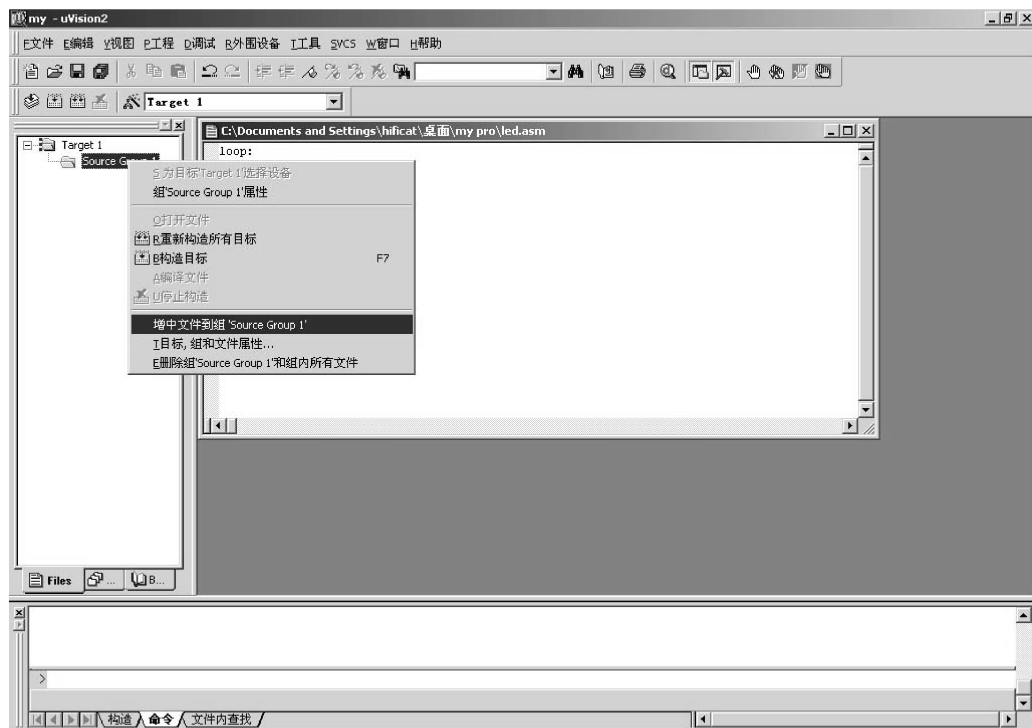


图 8-6　操作界面

接下来，我们要为源程序做一项编译工作，即产生目标文件，我们要将该文件烧入到AT89C51单片机芯片中去。在执行编译之前，我们需要进行一些设置，右击“Target 1”，在弹出菜单中选择“目标 Target 1 属性”选项，进入弹出菜单中的“输出”页，页面中有一项为“生成 HEX 文件”，我们在其选择框内打上勾，然后点击“确定”按钮完成设置。现在，我们只要按一下快捷键“F7”，就可以完成编译工作了，这时你会在“led. asm”文件所在目录下发现一个名为“my. hex”的文件，这就是我们所用来完成烧写芯片工作时使用到的目标程序文件，该文件为十六进制文件。

编程完成后，自然要用仿真器来验证一下程序是否正确。在使用仿真器之前，我们还需要手动设置一些相关参数，同样是在“目标 Target 1 属性”选项，进入“目标”页面，将晶振频率设置为11.0592MHz，因为仿真器使用的频率值为11.0592MHz，如图8-7所示。



图 8-7 “目标‘Target 1’属性”界面

进入“调试”页后，选择使用“Keil Monitor-51 Driver”硬件仿真器，点击其右边的“设置”按钮，进行仿真器的串口通信设置，如果仿真器串口线插在计算机的COM1口上，则选择“COM1”，因为笔者使用时是插在COM2口上，所以选择了“COM2”，将波特率设置为“38400”，点击“OK”按钮后，我们在“启动时加载程序”的复选框打勾，将页面内的“恢复调试设置”按需选择即可，在此将“断点”、“工具栏”、“浏览点 &PA”、“存储器显示”这几项打上勾。详细的设置，如图8-8所示。

我们已经将所有的设置完成了，让我们来看看成果吧。首先，点击 Keil 软件“调试”菜单中的“开始/停止调试”项，或者也可以按键盘快捷键“Ctrl + F5”。如屏幕左下角出现如图8-9所示的样子，则表示仿真器连接成功，“Monitor-51 V3.4”是软件版本号。



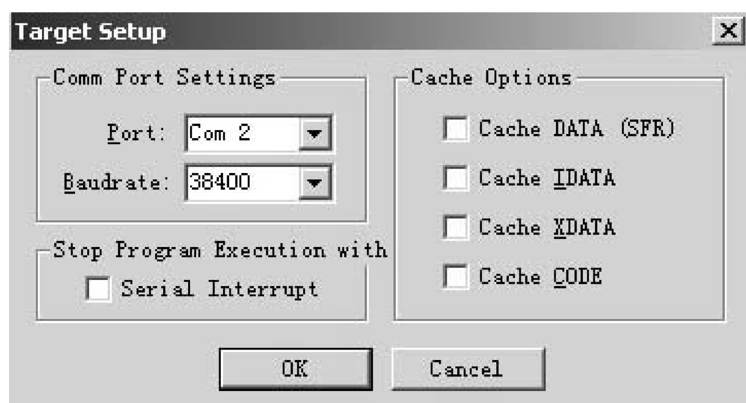


图 8-8　调试界面

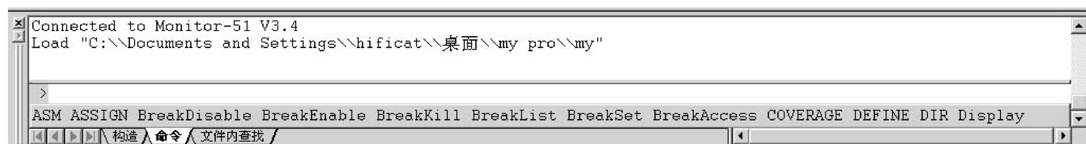


图 8-9　仿真器连接成功界面

然后，再选择“调试”菜单中的“运行到”按钮，或使用键盘快捷键“F5”，这样仿真器才真正地起到仿真的作用了，你会发现实验板上的第一个 LED 亮了，这正是我们所想要的结果，如图 8-10 所示。

至此，我们已经完成了程序调试工作及硬件的仿真。但是，我们还需要做一件事，就是断开连接，如我们在生活中打完电话一定要挂机一样，断开连接的操作非常简单，首先按一

下仿真器硬件电路板上的一个复位按钮（见仿真器图片中左上角那个按钮）。然后，按“开始/停止调试”按钮，即我们刚才用来连接时按的那个按钮。至此，仿真工作全部结束，现在对仿真器的使用应该有了大致地了解。

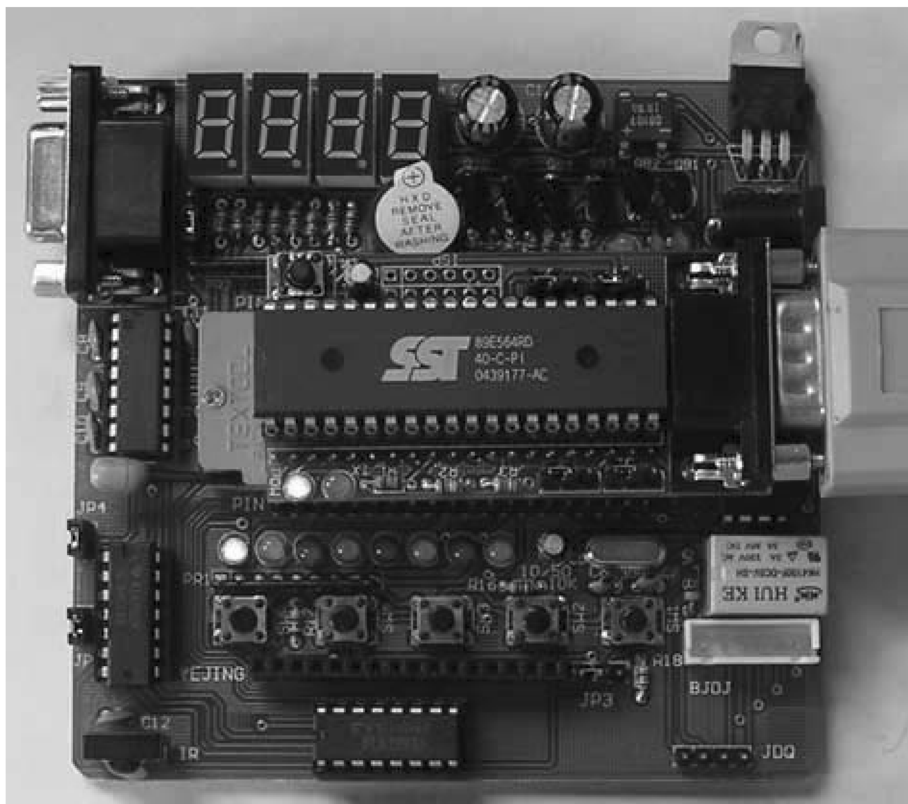


图 8-10　实验板

8.4　芯片烧写操作指南

我们现在已经完成了软件程序的编制及仿真工作，然后进行最后一道工序，即程序定形后，如何将其烧到单片机芯片中去。我们以 Atmel 公司的 AT89C51 芯片为例，也可以使用最新的芯片如 AT89S51、AT89S52 芯片等进行烧写实验。

首先，我们将串口线从仿真器上拔下，然后插在 A51 编程器上，同时插上 USB 线，如图 8-11 所示。

将光盘上的“A51 经济型编程器软件”文件夹全部复制到你的计算机硬盘上，并将其目录下所有文件的“只读”属性去掉，具体操作为全选所有文件，在文件属性中将其“只读”项前面复选框内的勾去掉即可。现在，我们打开“编程器 . exe”，进入程序界面，同样编程器在第一次使用前也需要手动设置一些参数，进入“设置”项，根据您编程器所插的 COM 口号，设置好 COM 口序号，波特率设置为 28800，在图 8-11 中笔者所插的 COM 口为 COM2，将界面左上角的芯片类型设为 AT89C51，如果您使用的芯片是 AT89S51，那么将界面左上角的芯片类型设为 AT89S51，设置步骤如图 8-12 所示。

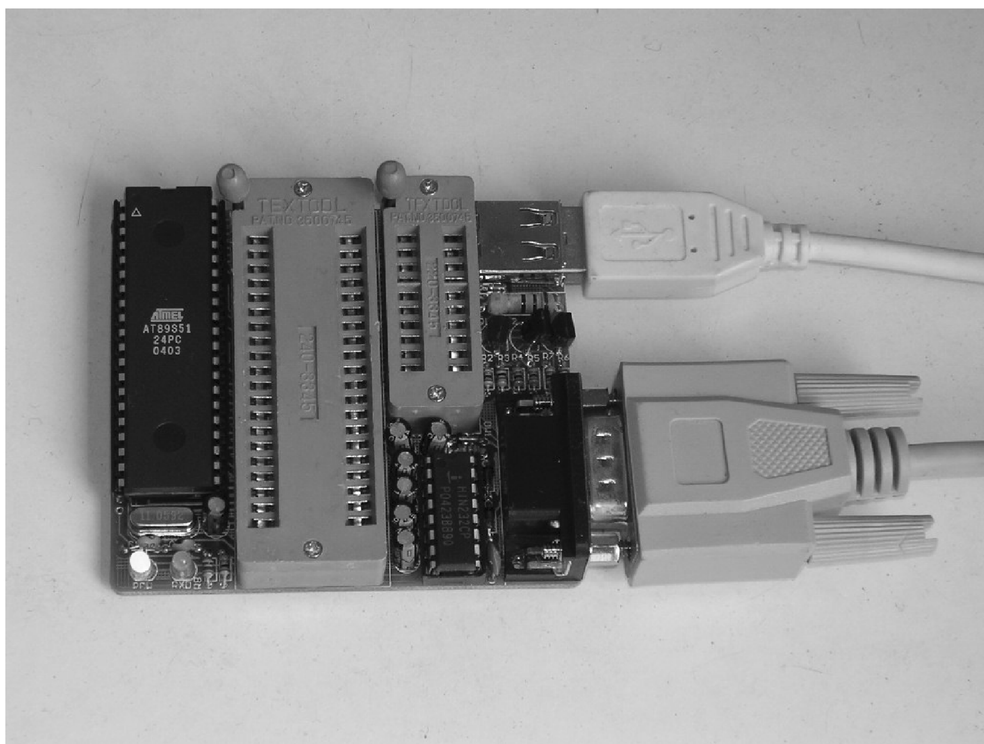


图 8-11　芯片烧写操作步骤



图 8-12　设置步骤

设置完成后，我们将要烧写的程序文件调进来，执行“打开文件操作”，找到刚才已经准备好的“my. hex”文件，选中“打开”即可。然后，我们插入要烧写的 AT89C51 芯片，如图 8-13 所示。

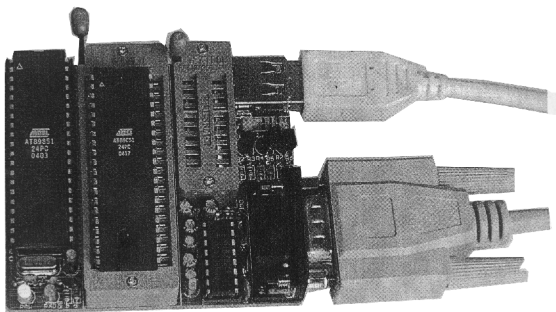


图 8-13　烧写程序文件步骤

首先，我们执行“擦除器件”操作，差不多 1s 即可完成芯片的擦除工作，然后用鼠标点一下“写器件”按钮，即大功告成。至此，我们已经完成了从软件编写，仿真，直到烧写芯片的全部步骤。最后看看我们的成果，将刚才烧写好的 AT89C51 芯片插在实验板上，并接上 USB 线，看看板上的第一个 LED 是不是亮了，如果板上的第一个 LED 点亮，说明已经脱离了仿真器而使用的是单片机芯片。

笔者写到这里，整个实验和操作步骤已经全部完成，虽然这是一个很简单的实验，但很多复杂的例子都是基于各种简单的原理之上。通过这个实例的学习，给大家一个感性的认识，最重要的是能够提供给大家一个实验的硬件环境以及软硬件相结合的实践描述，以增加单片机初学者的实践动手能力。我们提供的实验板上的资源非常丰富，可以做流水灯、数码管、蜂鸣器、键盘、继电器控制、IIC 总线通信等实验，光盘上也都配有例程、实验中的一些视频操作录像及编程器、仿真器的全部驱动程序，以供大家方便学习，实验中的一些视频录像请见光盘“单片机实验视频录像”下的视频文件。因此，你只要有一台计算机就可以进行学习、开发了，相信你只要发挥你的想象，一定可以使单片机发挥出更大的潜力。

MCS-51 单片机引脚说明

51 系列单片机 8031、8051 及 89C51/89S51 均采用 40Pin 封装的双列直接 DIP 结构。图 8-14 是它的引脚配置：在 40 个引脚中，有正电源和地线两根，外置石英振荡器的时钟线两根，4 组 8 位共 32 个 I/O 口，中断口线与 P3 口线复用。我们对这些引脚的功能加以说明：

1）Pin20：接地脚。

2）Pin40：正电源脚，工作时，接 +5V 电源。

3）Pin19：时钟 XTAL1 脚，片内振荡电路的输入端。

4）Pin18：时钟 XTAL2 脚，片内振荡电路的输出端。

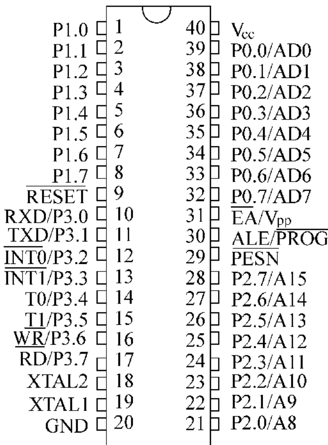


图 8-14　引脚配置

8051 的时钟有两种方式：一种是片内时钟振荡方式，但需在 18 和 19 脚外接石英晶体（2～12MHz）和振荡电容，振荡电容的值一般取 10～30pF；另一种是外部时钟方式，即将 XTAL1 接地，外部时钟信号从 XTAL2 脚输入，如图 8-

15 所示。

输入输出（I/O）引脚：Pin39 ~ Pin32 为

P0. 0 ~ P0. 7 输入输出脚。

Pin1 ~ Pin8 为 P1. 0 ~ P1. 7 输入输出脚。

Pin21 ~ Pin28 为 P2. 0 ~ P2. 7 输入输出脚。

Pin10 ~ Pin17 为 P3. 0 ~ P3. 7 输入输出脚。

- Pin9：RESET/Vpd 复位信号复用脚，当 8051 通电，时钟电路开始工作，在 RESET 引脚上出现 24 个时钟周期以上的高电平，系统即初始复位。

8051 的复位方式可以是自动复位，也可以是手动复位，如图 8-16 所示。此外，RESET/Vpd 还是一复用脚，VCC 掉电期间，此脚可接上备用电源，以保证单片机内部 RAM 的数据不丢失。

- Pin30：ALE 当访问外部程序器时，ALE（地址锁存）的输出用于锁存地址的低位字节。而访问内部程序存储器时，ALE 端将有一个 1/6 时钟频率的正脉冲信号，这个信号可以用于识别单片机是否工作，也可以当作一个时钟向外输出。如果单片机内存是 EPROM，在编程其间，将用于输入编程脉冲。

- Pin29：当访问外部程序存储器时，此脚输出负脉冲选通信号，PC 的 16 位地址数据将出现在 P0 和 P2 口上，外部程序存储器则把指令数据放到 P0 口上，由 CPU 读入并执行。

- Pin31：EA/Vpp 程序存储器的内外部选通线，8051 和 8751 单片机，内置有 4kB 的程序存储器，当 EA 为高电平并且程序地址小于 4kB 时，读取内部程序存储器指令数据，而超过 4kB 地址则读取外部指令数据。如 EA 为低电平，则不管地址大小，一律读取外部程序存储器指令。

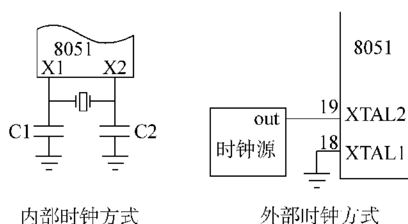


图 8-15　51 单片机振荡电路

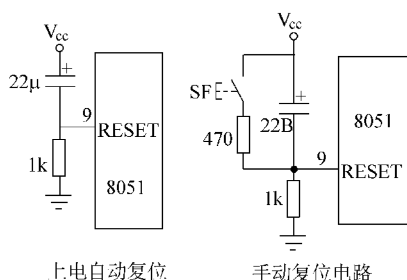


图 8-16　复位电路

8.5　增强型 51 单片机实验板常见问题解答

如果您是一个初学者，应仔细地阅读以下内容，并反复地校对自己可能会出错误的地方。

1. 为何提示芯片烧写成功并目标板运行时，但增强型 51 单片机实验板却没有反应？

答：检查卡座上的芯片是否为 AT89S51？或者根本没有放芯片，当然不能运行。也有可能是系统没有正确复位，可以按一下系统的复位键看看。

2. 在仿真状态下，为何总是不能联机？

答：在仿真状态下，必须使用我们光盘中的 keil uv2 软件，并且注意这个软件必须安装在 C 盘的根目录下，不能安装在其他的目录下！建立文件的时候，也不要保存在其他的目录中，更不要使用中文的文件名和文件夹。

3. 插上外接电源或 USB 供电线，实验板电源指示灯不亮？

答：请检查实验板供电类型选择跳线 JP1，可以试试左右位置互换一下。

4. 为何有些程序在烧写方式运行时可以，而仿真方式却不行？

答：在仿真状态下，会占用部分的系统资源，不能做到 100% 完全真实的仿真，事实上，仿真的概念也就是“模拟”，如果您对结果有疑问，那么请将程序 hex 文件烧入芯片来运行。

5. 如果我想用 89C2051 单片机怎么办？

答：由于 2051 和 S51 相比，只是少了 P0 口和 P2 口，其编程方法、指令都是一样的，所以你完全可以用 S51 代替 2051 来做实验，编程时避开 P0、P2 口即可。如果必须用 2051，那么我们将用优惠的价格供应其他编程器。

6. 随机配的 89S51 和书上的 89C51 是一样的么？

答：89S51 是 89C51 的升级版本，它们的指令格式完全是兼容的，也完全可以相互替代，并且价格更低，寿命更长。89S51 增加了看门狗功能和 isp 下载功能。

7. 是否可以带电插拔串口？

答：不要带电插拔串口和并口，下面的操作顺序可以避免带电插拔串口。

仿真联机正确的顺序为：用串口线将仿真器与计算机相连→将仿真器插到实验板上→给增强型 51 单片机实验板插上 USB 供电线或 9 ~ 12 外接电源。

仿真脱机正确顺序：拔下 USB 供电线或 9 ~ 12 外接电源→拔下仿真器→取下串口线。

总之，在电源开关处于关闭状态的情况下，插拔串口不会有任何危险。

8. 89S51 芯片和仿真芯片是否可以在电源接通的情况下插拔？

答：89S51 芯片在不处于写入状态时，随时都可以插拔！和电源状态无关。

仿真芯片在插拔之前请先按一下系统的硬件复位按钮，按完之后随时都可以插拔！和电源状态无关。

9. 开机后，如果插入空白的 AT89S51 芯片，P0 和 P2 的指示灯都会亮起来，是否正常？

答：正常，AT89S5* 系列芯片在空白状态下会输出闪动的低电平，这是它的一个固有特性，写入程序后此特性消失。

10. A51 编程器软件提示栏没有出现编程器就绪的字样？

答：检查串口线是否与计算机连接，同时配置好正确的 COM 口序号及相关参数，设置更改后，重启程序后生效；在串口线已连接，并在软件配置正确的情况下，可以通过拔插 A51 编程器的 USB 供电线来激活通信程序，从而让编程器处于就绪状态。

第 9 章　单片机入门基础实例

9.1　点亮一个发光二极管

发光二极管，也叫做 LED，是一种常用的指示器件，例如电源指示、工作指示等。即便你不怎么留心，恐怕还是会在不少场合见到过，如各种充电器，它们用亮或者灭告诉你电源是否已经接通，用颜色的变化告诉你电池是否已经充满。再如有不少设备，往往采用发光二极管的闪烁来表示系统正常工作。因为它的控制比较简单有趣，所以我们的实验就选择从这里开始。

9.1.1　实现方法

首先我们需要知道如何让一个发光二极管工作。发光二极管种类很多，如图 9-1 所示的是几种直径为 3mm 的普通亮度发光二极管，其图形符号如图 9-2 所示，当在它的 A 和 K 两个电极加上合适的电压时，它就会亮起来。说“合适的电压”，是因为不同的发光二极管工作电压并不相同，一般是在 1.6～2.8V 之间，而工作电流则一般在 2～30mA 之间，但是实际工作中选择的范围一般是 4～10mA 之间。这里之所以要说这些参数，实际是为了解释 LED 上串接电阻大小的选择。图 9-3 所示是实验板上与 LED 控制相关部分的电路，我们可以看到 LED 上串接的电阻是 1kΩ，如果此时 LED 上的电压是 2.0V，那么此时通过 LED 的电流则为（5V－2V）/1000Ω＝3mA。如果需要提高亮度，一般会把电流控制在 10mA 左右，则此时电阻应该选择（5V－2V）/10mA＝300Ω，所以可以就近选择 330Ω。



图 9-1　发光二极管实物图

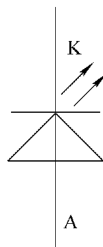


图 9-2　发光二极管的图形符号

电路已经确定，然后连接到单片机的 I/O 口上（见图 9-3），我们可以看到 LED 的 A 极通过限流电阻连接到 V_{CC}，K 极连接到了单片机的 I/O 口。因此，要使 LED 发光，只需要把 I/O 口置成低电平即可。所以最终我们对 LED 的控制变成了对一个 I/O 口的控制，例如要点亮 LED1，就把 P1.0 设置成低电平，这就是实现方法。

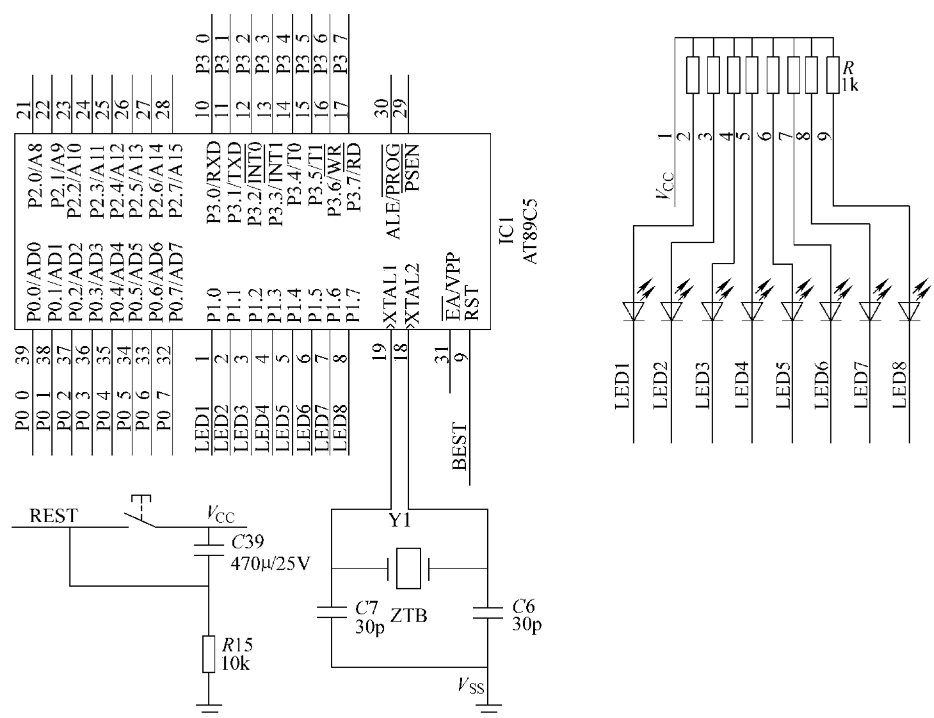


图 9-3　单片机控制 LED 显示的原理

9.1.2　源程序

```
01  #include <reg51.h>
02
03  sbit LED = P1^0;
04
05  void main (void)
06    {
07        LED  =  0;
08        while (1);
09    }
```

9.1.3　代码分析

序号 1：包含 51 单片机寄存器定义的头文件

序号 3：位定义 LED 为 P1.0

序号 5 ~ 9：main 程序

序号 7：使 P1.0 端口输出电平 0，点亮发光二极管

序号 8：循环等待

9.2 使发光二极管闪动

同常亮的发光二极管相比，闪动的方式可以更好地体现运行状态，这节我们就要使它闪动起来。

9.2.1 实现方法

我们已经知道，要使图 9-3 中的 LED1 发光，只要把 P1.0 置成低电平就可以了，反之，把 P1.0 置成高电平就可以使 LED1 灭掉。而要想让他闪动起来，即让亮和灭在一段时间内交替出现。实现的方法就是使 P1.0 在每隔一段时间轮流出现高低电平。这里留心一下“一段时间”，因为单片机的程序执行速度很快，如果是在很短的时间内改变 P1.0 的状态，人眼是看不出来的，中间必须有个合适的延迟时间，所以一般闪动的延迟是在 300ms 左右，以这个频率闪烁不会令人觉得很紧张。

9.2.2 源程序

```
01  #include <reg51.h>
02
03  sbit LED = P1^0;
04
05  void Delay ()
06   {
07      unsigned char i, j;
08         for (i=0; i<255; i++)
09            for (j=0; j<255; j++);
10  }
11
12  void main ()
13   {
14    while (1)
15     {
16      LED = 0;
17      Delay ();
18      LED = 1;
19      Delay ();
20     }
21  }
```

9.2.3 代码分析

序号 1：包含 51 单片机寄存器定义的头文件

序号 3：位定义 LED 为 I/O 口 P1.0

序号 5 ~10：一个延时函数，具体延长的时间和使用的晶体相关

序号 7：定义两个无符号变量 i，j

序号 8 ~9：通过 i，j 的自加嵌套循环执行，达到延时的目的

序号 12 ~21：main 函数

序号 14：进入主程序的 while 循环

序号 16：点亮 LED

序号 17：调用延时程序

序号 18：熄灭 LED

序号 19：调用延时程序

9.2.4 深入了解

通过修改上面的延时函数的数据，我们可以改变发光二极管的闪烁频率，应该说，到此为止我们已经实现了我们需要的功能，但是还有一些问题，需要进一步了解。

由于位变量只有两个状态，0 或者 1，同时 C 语言里有种取反的运算，所以上面的置 1 和清零也可以通过一个句子来实现，LED =！LED，意思就是 LED 的值取它相反的值，这样，上面程序 16 ~19 就可以用下面两句程序替换：

LED = ！LED；

Delay（）；

同时，我们再来考虑另外一个问题，由于我们在主程序 while 循环里，为了闪烁的效果，两次调用了 Delay（） 函数（现在的程序，基本上什么事情也不做），所以，最后出来的效果还是很令人满意。但是在一个真正实用的系统里，while 循环里会有很多事情要做，例如上面的程序里，在 19 和 20 行之间还有很多程序要执行，如果后面部分程序比较多的话，我们会发现，闪烁的亮灭时间不再均匀，同时随着程序执行顺序的问题，不同次的闪动周期也可能差别很大。问题还不止一个，我们已经知道 while 循环要做很多事情，因此整个执行周期本来可能就不小，在循环里两次调用了延时函数，即使改成用取反的办法来实现，也还是需要调用一次，这时假设延迟函数是 500ms，那么所有由这个 while 循环实现的工作是不是无论如何也没有办法执行得比 500ms 更快了？显然这里延时函数的调用是很要命的缺点。

正是由于上述的原因，在真正实用化的时候，这个程序的思路需要修改，最需要引入的概念是“定时中断”，一般来讲，每隔一定时间做的事情，都可以由定时中断来处理。

9.3 流水灯

在试验板上制作的流水灯与你在街上看到的不一样，因为我们在实验板上只能用 LED 来模拟 8 个灯，但是它们的控制原理则是一样的，如果把这 8 个 LED 替换成 8 个继电器，然后再去控制 8 个彩灯甚至是 8 组彩灯时，结果就是真正的流水灯了。

首先，来看看我们希望实现的目标：使 8 个编号为 1 ~8 的 LED 从 LED1 开始亮起，每次只点亮一个，并按顺序往 LED8 移动，结束后再次从头开始。实际上就是在程序开始执行之后，使程序一直在“复位状态”到“状态 8”之间按顺序执行见表 9-1。

表 9-1　流水灯状态表

LED 序号	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8
对应的 I/O 口	P1. 0	P1. 1	P1. 2	P1. 3	P1. 4	P1. 5	P1. 6	P1. 7
复位状态								
状态 1								
状态 2								
状态 3								
状态 4								
状态 5								
状态 6								
状态 7								
状态 8								

9. 3. 1　实现方法

前面我们已经知道了操纵一个 LED 的方法，要实现这个程序，关键是了解整个过程。我们先来比较一下“复位状态”和“状态 1”的区别，有什么地方不一样呢？就是 LED1 被点亮了，所以从“复位状态”到“状态 1”，需要完成的操作是“点亮 LED1”。然后从“状态 1”到“状态 2”，我们很容易发现，区别的地方有两个，即 LED2 亮了，LED1 却灭了。在这一步，需要做的是“点亮 LED2”和“熄灭 LED1”。按照这样的过程，其他步骤需要完成的任务也不难看出来，列出的流程如图 9-4 所示。

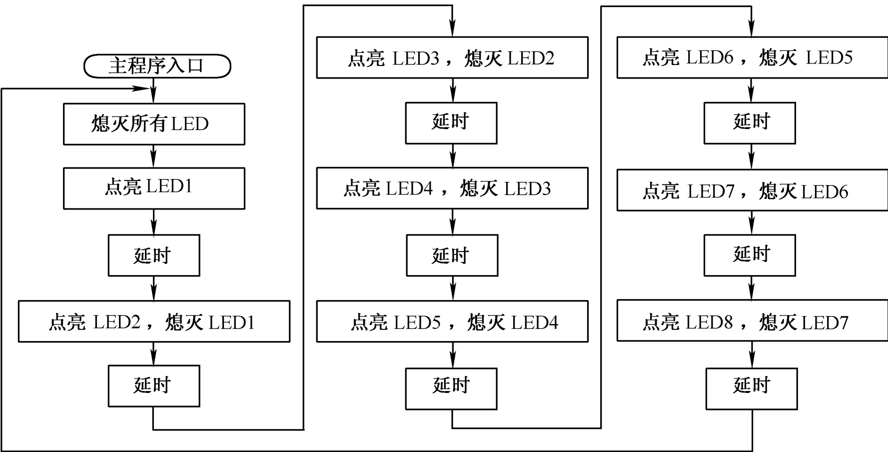


图 9-4　流水灯程序流程

9.3.2 源程序

```
#include <reg51.h>

sbit LED1 = P1^0;
sbit LED2 = P1^1;
sbit LED3 = P1^2;
sbit LED4 = P1^3;
sbit LED5 = P1^4;
sbit LED6 = P1^5;
sbit LED7 = P1^6;
sbit LED8 = P1^7;

void Delay ()
 {
    unsigned char i, j;
      for (i=0; i<255; i++)
        for (j=0; j<255; j++);
}

void main ()
 {
    while (1)
     {
      P1 = 0xff;
      LED1 = 0;
      Delay ();
      LED2 = 0;
      LED1 = 1;
      Delay ();
      LED3 = 0;
      LED2 = 1;
      Delay ();
      LED4 = 0;
      LED3 = 1;
      Delay ();
      LED5 = 0;
      LED4 = 1;
      Delay ();
```

```
39          LED6 = 0;
40          LED5 = 1;
41          Delay ();
42          LED7 = 0;
43          LED6 = 1;
44          Delay ();
45          LED8 = 0;
46          LED7 = 1;
47          Delay ();
48        }
49    }
```

9.3.3 代码分析

序号 1：包含 51 单片机寄存器定义的头文件

序号 3 ~ 10：位定义 LED1 ~ LED8 分别对应 I/O 口 P1.0 ~ P1.7

序号 12 ~ 18：一个延时函数，具体延长的时间和使用的晶体相关

序号 20：main 函数

序号 24：P1 口全部置 1，即所有 LED 熄灭

序号 25：LED1 点亮

序号 26：调用延时程序

序号 27：LED2 点亮

序号 28：LED1 熄灭

序号 29：调用延时程序

序号 30 ~ 47：按照流程操作 LED

9.3.4 深入了解

相信到这里，你已经看到发光二极管在闪烁了，如果仅仅是对这个程序要求的功能来讲，已经没有什么可以挑剔的了，而且通过对这个程序实现的分析，相信凭你自己的能力，也一定可以写出另外花样的流水灯了。但是你也许看到过别人用更加简单的代码来实现这个流水灯的功能，那就意味着，这个程序还可以按照另外的思路来做。例如如下的代码：

```
#include <reg51.h>

void Delay ()
{
    unsigned char i, j;
    for (i=0; i<255; i++)
        for (j=0; j<255; j++);
}
```

```
void main ( )
{
    unsigned char i;
    unsigned char temp;

    P1 = 0xff;                  //P1 口全部置 1，熄灭所有 LED

    while (1)
     {
        temp = 0x01;            //赋初值给 temp，只有 1 位为 1
        for (i = 0; i < 8; i++)
         {
          P1 = ~temp;           //将 temp 取反后送 P1 口输出
          Delay ( );            //调用延时函数
          temp = temp << 1;     //temp 中的数据左移 1 位
        }
    }
}
```

这个程序实现的功能和 9.3.2 中程序实现的功能完全一致。其最大的不同是在思路上，它已经不再单独的来看某个 LED 了，所以程序中也不再定义 LED 的连接，可以认为它将 8 个 LED 合成一个整体来考虑，也可以说，其实它就只关心 P1 口，因为 LED1 ~ LED8 本来就是和 P1.0 ~ 1.7 对应的。所以程序的目的是在 P1 口上实现这样的功能：从 P1.0 ~ P1.7，按顺序循环使某个口线为低电平，同时其他口线为高电平。再进一步，现在 P1 口的数据和变量 temp 是对应起来的，所以最终对 8 个 LED 的操作变成了对变量 temp 的操作。我们用流程来看一下 temp 的变化，也就明白整个过程了，见表 9-2。

表 9-2　temp 的变化过程

	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7
temp	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
~ temp	0xfe	0xfd	0xfb	0xf7	0xef	0xdf	0xbf	0x7f

9.4 按键操作

按键是单片机系统中常用的信息输入部件，同时也是人机对话中不可缺少的输入设备，其外形如图 9-5 所示。在和单片机构成系统的时候，按键通常有两种接法，一种叫做独立式按键，另外一种叫做行列式或者是扫描式按键，由于受实验板资源的限制，我们在这里只学习独立式的按键电路。在这个实验里，我们需要用 S1 和 S2 来控制 LED1 的亮和灭。

9.4.1 实现方法

首先我们来了解一下按键的结构。一般的按键从实物来看，是个四端口器件，但实际它

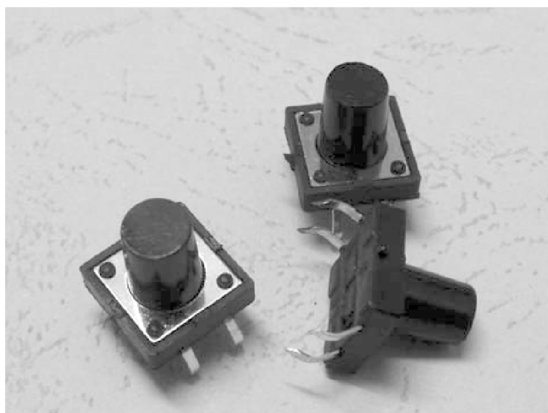


图 9-5　某种按键外形

是个二端口器件，参照图 9-6 中的 S1 ~ S4 我们就不难明白，在按下按键之前，两个触点之间是不导通的，按下的时候就导通，通过外部电路的不同接法，就可以使其中一个端口在按下和不按下的时候产生电平变化，而单片机正是通过检测到这种变化来完成对按键输入信息的获得。那么我们不妨来看看原理图所示的按键的状态变化。我们知道单片机的 P3 口有内部上拉电阻，所以在按键按下之前，S1 对应的端口 P3. 5 保持在高电平状态，当按键按下时，P3. 5 通过 S1 接到 V_{SS}，这个时候就是低电平。所以，我们要想在程序里检测到是否有按键按下，关键就是检查对应端口的状态变化，这就是单片机系统中的按键编程的原理。所以针对我们的程序设计目的，我们的方法就是不断地检测 P3. 5 和 P3. 4，然后根据检测结果

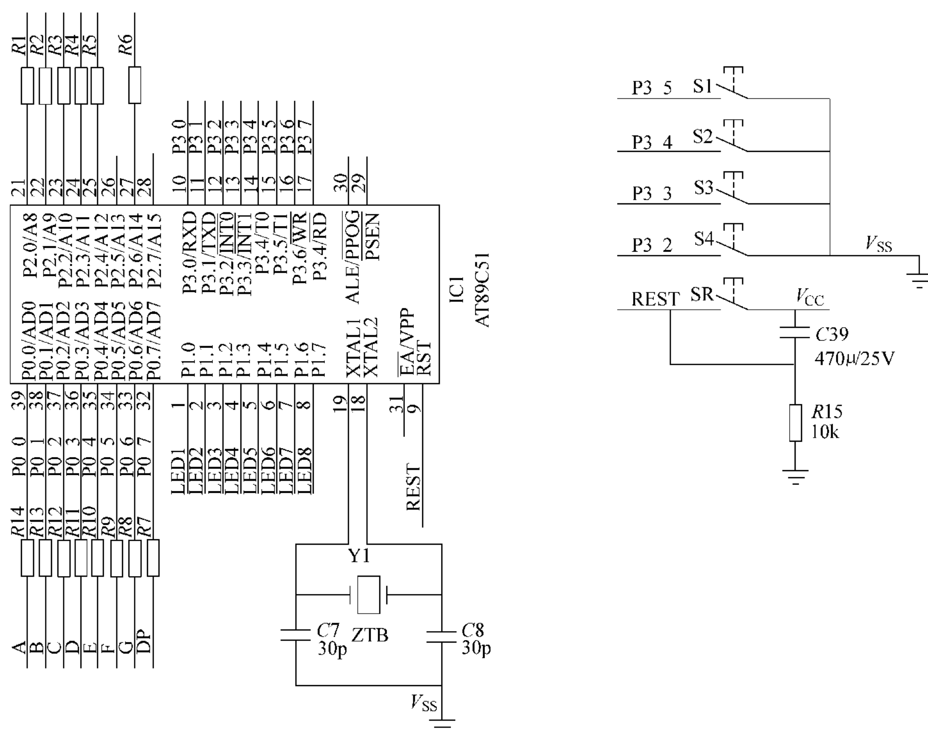


图 9-6　按键部分相关原理

控制 LED1。

9.4.2 源程序

```
01 #include <reg51.h>
02
03 sbit S1 = P3^5;
04 sbit S2 = P3^4;
05 sbit LED1 = P1^0;
06
07 void main ()
08  {
09      P3 = 0xff;
10
11      while (1)
12       {
13        if (S1 = =0) LED1 = 0;
14        if (S2 = =0) LED1 = 1;
15       }
16 }
```

9.4.3 代码分析

序号 1：包含 51 单片机寄存器定义的头文件
序号 3：位定义 S1 为 P3.5
序号 4：位定义 S2 为 P3.4
序号 5：位定义 LED1 为 P1.0
序号 7～16：main 程序
序号 9：使 P3 口全部输出为高，在读入端口外部数据之前，必须置高该端口
序号 11～15：循环执行程序
序号 13：如果检测到 S1 按下，则点亮 LED1
序号 14：如果检测到 S2 按下，则熄灭 LED1

9.4.4 深入了解

上面的程序执行下来，效果很好，至少目前是如此，但是如果我们要用一个 S1 按键来控制 LED1 的开关，请想想，会出什么问题呢？

在实际应用中，要使用一个按键来控制某种功能的打开和关闭是很常见的。例如我们要求开/关一个发光二极管，开和关的动作可以用取反运算来代替，即亮和灭交替出现，现在我们把上面程序里的 13 和 14 两句替换成一个句子 if（SW1 = =0）LED1 = ！LED1，看看执行结果，看看是不是可以达到我们的目的。

结果怎么样？是不是发现 LED 的亮和灭的状态总是不确定的？其实我们早就知道单片

机的程序执行速度很快，所以按照现有程序，我们不难分析出，我们写的那句子 if（SW1 = =0）LED1 = ！LED1 在我们手按下按键这个时间里绝对不可能只执行一次，比如我们随便按一下按键，估计也有 200ms 以上的时间，而循环执行上面写的那个语句，执行周期恐怕不会大于 20μs，所以要想达到我们预期的效果，程序肯定不可以照这样写。

比较通行的解决办法有不少种，比如可以在检测到按键后使下次的按键检测在一段时间之后才开始，即采取延时；再比如，可以用统计重复按键的办法来确定一次按下的完整起止过程；还有比较方便的一种做法，就是在检测到按键后一直等着按键松开才执行相关操作，我们在这里就用这个办法来做。

```
#include <reg51.h>

sbit S1 = P3^5;
sbit S2 = P3^4;
sbit LED1 = P1^0;

void main ( )
{
  P3 = 0xff;

  while (1)
   {
    if (S1 = =0)            //如果 S1 按键按下则开始判断
     {
      if (S1 = =1)          //如果 S1 按键松开则执行 LED1 = ! LED1
       {
        LED1 = ! LED1;
       }
     }
   }
}
```

9.5 蜂鸣器的使用


图 9-7 蜂鸣器外形

在实验板的上部，4 个数码管的右下角，有一个形状如图 9-7 所示的部件，叫做蜂鸣器，也就是 BUZZER。它和普通扬声器相比，最重要的一个特点是只要按照极性要求加上合适的直流电压，就可以发出固有频率的声音，因此使用起来比扬声器简单，所以在一些报警要求不高的场合有所使用。还有一个很常见的用法是，当按下按键操作的时候，很多单片机构成的系统会发出声音提示，一般而言，也主要是由它来做的，但是它不能发出语音。虽然它有自己的固有

频率，但是它也可以被加以不同频率的方波，从而编制一些简单的音乐。

9.5.1 实现方法

从上面的描述中，很容易地得到一个结论，那就是蜂鸣器的控制和 LED 的控制对单片机而言是没有区别的，再加上如图 9-8 所示的晶体管的接法，更加使得其控制和第一小节 LED 的控制完全一致了：低电平有效，即 I/O 口输出“0”就响，反之则不响。

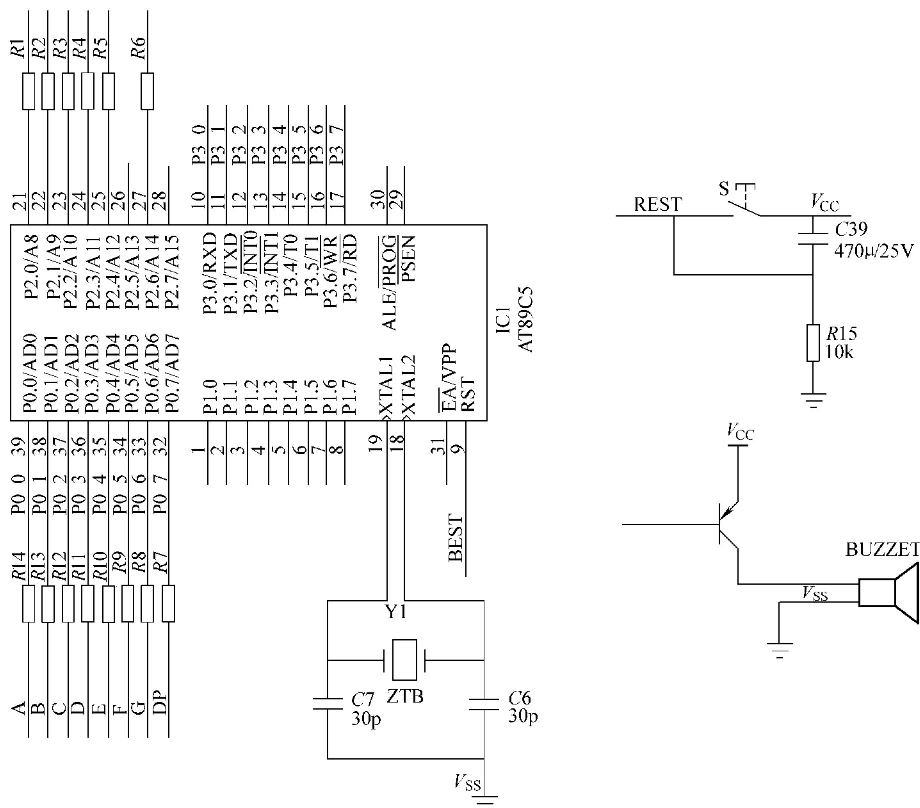


图 9-8 蜂鸣器的相关部分原理

9.5.2 源程序

```
#include <reg51. h>

sbit BUZZER = P1^0;

void main (void)
  {
      BUZZER = 0;
      while (1);
  }
```

9.5.3 代码分析

序号 1：包含 51 单片机寄存器定义的头文件；
序号 3：位定义 BUZZER 为 P1.0；
序号 5 ~9：main 程序；
序号 7：使 P1.0 端口输出电平 0，蜂鸣器发声；
序号 8：循环等待。

9.6 数码管的使用

在这个小节里，我们要实现一个每隔一定时间循环显示数字 0 ~9 的程序，不过之前我们先要了解一下它的结构和特点。

数码管作为一种应用十分普遍的显示器件，可以在各种各样的设备上见到，如图 9-9 所示的就是数字表头显示时候的效果图。它很适合用在对价格、亮度等条件比较敏感，同时基本上只要求显示数字量的时候，所以在数据显示，定时控制等场合用得很多。

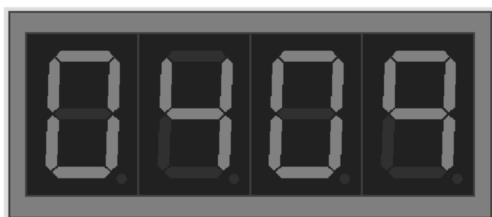


图 9-9　数字表头显示效果

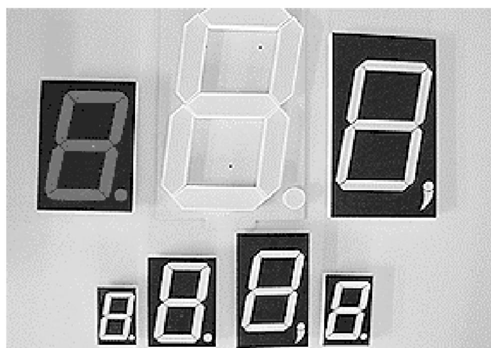


图 9-10　常见的数码管实物图

数码管也叫 LED 数码显示器，其实是由多个 LED 排列封装而成，图 9-10 给出了一些常见数码管的实物图，看到的数码管都是由 8 个 LED 组合而成的，当然也有其他类型的，其结构如图 9-11 所示，其中 7 个发光二极管排列成 8 字形，另外一个则是圆点状的，通常用来作为数据显示时的小数点使用。

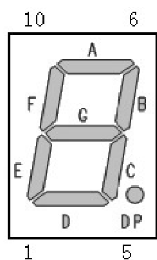


图 9-11　某类型数码管结构图

注：3 和 8 为公共引脚，A-7，B-6，C-4，D-2，E-1，F-9，G-10，DP-5。

由于驱动方式的差异，也就是对应在各个显示段是低电平还是高电平点亮，数码管又分成两种类型，即共阳极和共阴极数码管。所谓“共阳极”即是 8 个 LED 的阳极连接在一起组成公共端，同理“共阴极”则是 8 个 LED 的阴极连接在一起组成公共端，其内部 LED 的连接方式如图 9-12 所示。

我们已经有了可以显示数字的器件，但是离我们能够显示数据还是有一点距离的，比如我现在要数码管上显示 6，该怎么办呢？

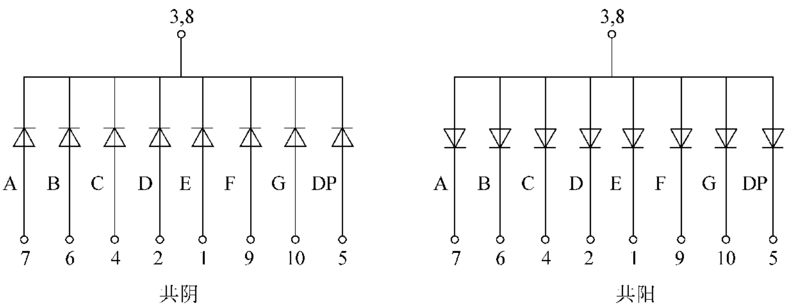


图 9-12　数码管的内部电路

9.6.1　实现方法

首先我们需要明白一个事情，数码管是不认识“6”的，当然也不认识其他数字，所以千万别说“给数码管写个 6 不就行了”。数字只是一种符号，对人来说是这样的，对单片机而言也是，单片机是通过 LED 把内部的结果用我们约定的方式显示出来，这个“约定”也包含了数字该如何在 LED 上显示的方法。比如我们需要显示的数字 0 ~ 9 如图 9-13 所示，并且假设我们使用共阴极数码管，然后我们对照图 9-14 来看看“6”是如何显示出来的。首先对数码管而言，我们要想显示数字“6”，可以发现有如下一些段是需要点亮的，即 A、C、D、E、F、G，对应到单片机的 I/O 口，除去 P0. 1 和 P0. 7 清零之外，其他的端口都要置成“1”。如果在程序里，我们是通过先查表，然后送出去来实现段码显示的，并且高低位刚好对应起来，那么我们可以列出这样的段码对应关系见表 9-3。

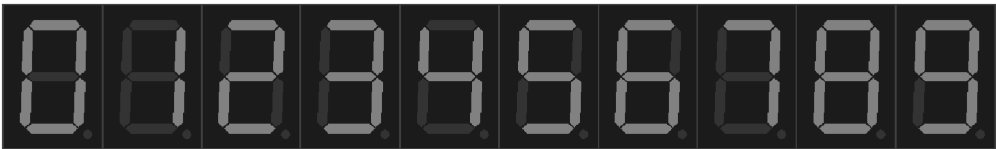


图 9-13　数码管数字显示效果图

表 9-3　数码管显示数字“6”的段码表

段名称	DP	G	F	E	D	C	B	A	对应段码
对应引脚	P0. 7	P0. 6	P0. 5	P0. 4	P0. 3	P0. 2	P0. 1	P0. 0	
数字 6	0	1	1	1	1	1	0	1	0x7d

参照上面的过程，可以列出共阴和共阳数码管 0 ~ 9 10 个数字的段码表，见表 9-4。在不改变硬件对应关系的前提下，段码表可以通用。

表 9-4　共阴、共阳数码管段码表

数字	0	1	2	3	4	5	6	7	8	9
共阴	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F
共阳	0xC0	0xF9	0xA4	0xB0	0x99	0x92	0x82	0xF8	0x80	0x90

现在我们已经了解了整个显示过程，所以我们也就有了写程序的思路：程序中应该有一个变量，每隔一定时间在 0 ~ 9 之间变化，然后按照这个数据去查找段码表，把查到的数据送到 P0 口。

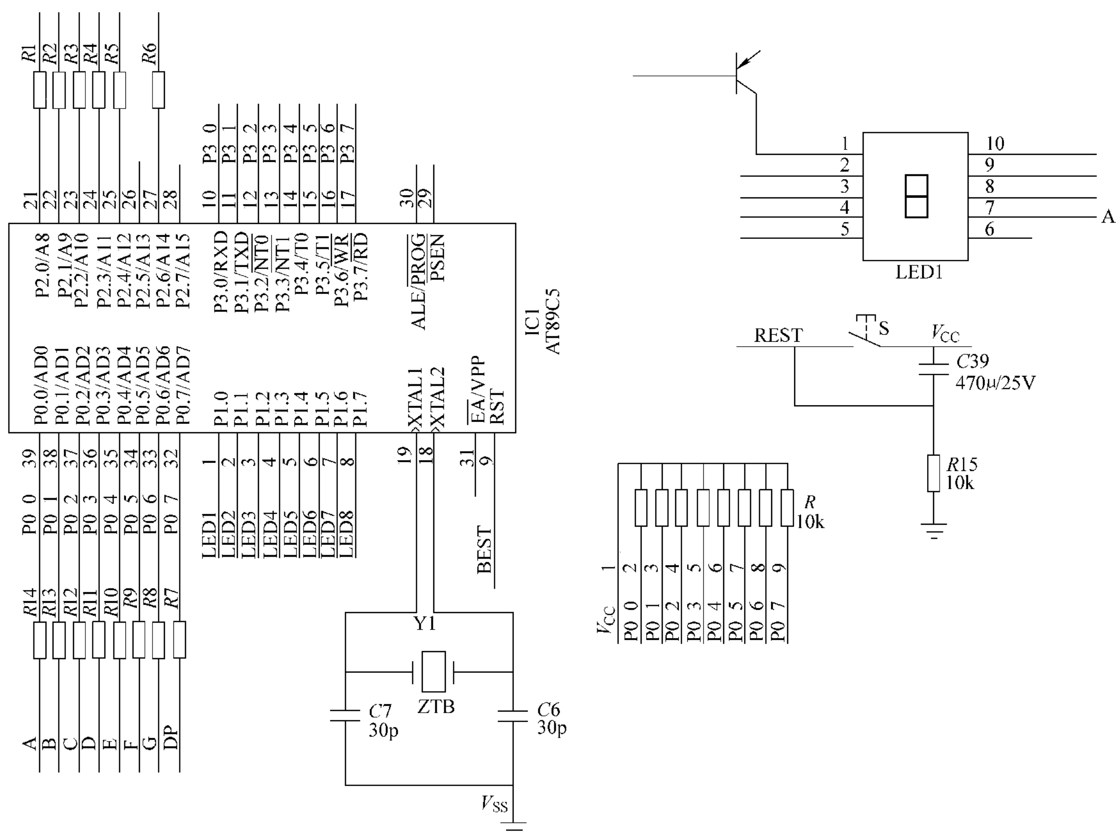


图 9-14　数码管显示相关部分电路

9.6.2　源程序

```
#include <reg51.h>

unsigned char code Tab[ ] = { 0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,
                              0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
sbit LED1 = P2^0;

void Delay( )
{
    unsigned char i,j;
        for(i=0;i<255;i++)
            for(j=0;j<255;j++);
}

void main( )
```

```
15    {
16        unsigned char i  = 0;
17        LED1  = 0;
18        while(1)
19        {
20            P0 = Tab[i];
21            Delay();
22            i++;
23            if(i>9) i =0;
24        }
25    }
```

9.6.3 代码分析

序号 1：包含 51 单片机寄存器定义的头文件

序号 3 ~4：定义共阳数码管的段码表

序号 5：位定义数码管 LED1 的公共控制端

序号 7 ~12：延时函数

序号 14：main 函数

序号 16：定义内部计数用的局部变量

序号 17：使数码管 LED1 公共端上电

序号 18：调循环程序

序号 20：从 P0 口输出 i 对应的段码

序号 21：调用延时程序

序号 22：i 自加 1

序号 23：对 i 进行处理，使其数值在 0 ~9 的范围内

9.6.4 深入了解

看到数字的跳跃，对任何一个喜爱单片机的朋友来说，相信这会是件很令人愉快的事情，尽管我们要说的东西还有一点，甚至还是有点麻烦的。

实验板上有 4 个数码管，并且他们的数据端口都是连接在 P0 口，如果我们在程序的开始处把 4 个数码管的显示都打开，那么相同的数据就会同时出现在这 4 个数码管上，但是假如我们需要在这 4 个数码管上分别显示 4 个数字，比如分别是 1、2、3、4 呢？这里就有了个“动态扫描显示”的说法，其实就是分时显示。先看下面的程序：

```
#include <reg51.h>

unsigned char code Tab[] = { 0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,
                             0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
sbit LED1  = P2^0;
sbit LED2  = P2^1;
```

```
07 sbit LED3  = P2^2;
08 sbit LED4  = P2^3;
09
10 void Delay( )
11 {
12       unsigned char i;
13              for( i =0;i <255;i + + );
14 }
15
16 void main( )
17 {
18       while(1)
19       {
20       P0 = Tab[1];
21       LED1  = 0;              //开 LED1 显示
22       Delay( );
23       LED1  = 1;              //关 LED1 显示
24
25       P0 = Tab[2];
26       LED2  = 0;              //开 LED2 显示
27       Delay( );
28       LED2  = 1;              //关 LED2 显示
29
30       P0 = Tab[3];
31       LED3  = 0;              //开 LED3 显示
32       Delay( );
33       LED3  = 1;              //关 LED3 显示
34
35       P0 = Tab[4];
36       LED4  = 0;              //开 LED4 显示
37       Delay( );
38       LED4  = 1;              //关 LED4 显示
39    }
40 }
```

在上述程序的 20 ~ 38 行，我们可以看到，显示过程是先把段码送出去，然后点亮对应的数码管，过段时间之后关闭，再开始另外一个，这里的延迟时间是以显示效果来定的。总之所谓动态扫描就是个分时显示的过程，只有这样才有可能共用数据端口，同时如果实际调试的时候显示效果不理想，那么要注意以下几点：看到数码管显示闪烁，是延迟过长，或者说扫描速度不够快；如果亮度不均匀，是因为每个数码管占到的时间片不一样；亮度很低，

是扫描过快了；感觉有重影，是中间切换过程没有处理好，应该是先关闭显示，然后送数据，然后再开显示。如果能注意以上说到的问题并解决好，相信写动态扫描的数码管显示程序就会比较简单了。

9.7 单片机继电器控制

在现代自动控制设备中，都存在一个电子电路（弱电）与电气电路（强电）的互相连接问题，一方面要使电子电路的控制信号能够控制电气电路的执行元件（如电动机、电磁铁、电灯等)，另一方面又要为电子电路和电气电路提供良好的电隔离，以保护电子电路和人身的安全。继电器便能完成这一桥梁作用。

9.7.1 继电器的工作原理与分类

继电器是一种电子控制元件，它具有控制系统（又称输入回路）和被控制系统（又称输出回路)，通常应用于自动控制电路中，它实际上是用较小的电流去控制较大电流的一种“自动开关”。在电路中起着自动调节、安全保护、转换电路等作用。在大多数的情况下，继电器就是一个电磁铁，这个电磁铁的衔铁可以闭合或断开一个或数个触点。当电磁铁的线圈中有电流通过时，衔铁被电磁铁吸引，因而就改变了触点的状态。继电器一般可以分为电磁式继电器、热敏干簧继电器、固态继电器等。本实验板上配置的继电器如图 9-15 所示。

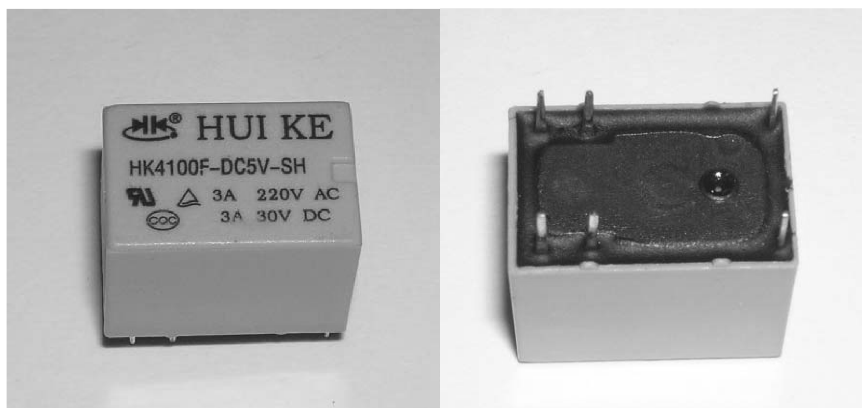


图 9-15 继电器实物图

电磁式继电器一般由铁心、线圈、衔铁、触点簧片等组成的。只要在线圈两端加上一定的电压，线圈中就会流过一定的电流，从而产生电磁效应，衔铁就会在电磁力的作用下克服返回弹簧的拉力吸向铁心，从而带动衔铁的动触点与静触点（常开触点）吸合。当线圈断电后，电磁的吸力也随之消失，衔铁就会在弹簧的反作用力下返回原来的位置，使动触点与原来的静触点（常闭触点）吸合。这样吸合、释放，从而达到了在电路中的导通、切断的目的。对于继电器的“常开、常闭”触点，可以这样来区分：继电器线圈未通电时处于断开状态的静触点，称为“常开触点”；处于接通状态的静触点称为“常闭触点”。

热敏干簧继电器是一种利用热敏磁性材料检测和控制温度的新型热敏开关。它由感温磁环、恒磁环、干簧管、导热安装片、塑料衬底及其他一些附件组成。热敏干簧继电器不用线

圈励磁，而由恒磁环产生的磁力驱动开关动作。恒磁环能否向干簧管提供磁力是由感温磁环的温控特性决定的。

固态继电器是一种两个接线端为输入端，另两个接线端为输出端的四端元件，中间采用隔离元件实现输入输出的电隔离。

固态继电器按负载电源类型可分为交流型和直流型。按开关型式可分为常开型和常闭型。按隔离型式可分为混合型、变压器隔离型和光隔离型，以光隔离型为最多。

本小节以电磁继电器为例，介绍其用法。

9.7.2 继电器的控制电路

在单片机系统中继电器的控制一般通过一个晶体管来驱动，典型的驱动电路如图 9-16 所示。

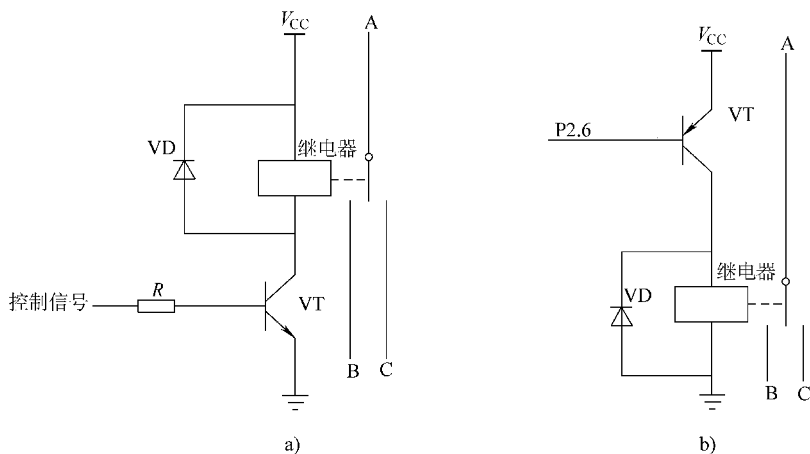


图 9-16　继电器的一般驱动电路

继电器电路中一般都要在继电器的线圈两端加一个二极管以吸收继电器线圈断电时产生的反电势，防止干扰。图 9-16 中的 AB 为常开触点，AC 为常闭触点。图 9-16a 中当控制信号为高电平时，继电器常开触点吸合（AB 导通）；当控制信号为低电平时，继电器常开触点断开，常闭触点吸合（AC 导通）。在图 9-16b 中控制信号极性正好与图 9-16a 相反，本书配套实验板上就是采用这个电路。

9.7.3 单片机控制继电器

从实验板原理图中，我们可以看到，单片机引脚 P2.6 与一个 PNP 型晶体管基极相连，经晶体管电流放大后，直接驱动继电器，继电器的开和关完全由晶体管的基极电平进行控制。当单片机 P2.6 口输出高电平，PNP 型晶体管截止，这时继电器不工作；反之为低电平时，PNP 型晶体管导通，继电器得电吸合。在掌握了继电器的工作原理和驱动方法后，我们来看一个单片机控制继电器的开合从而控制电灯的例子。实验板上的电路原理如图 9-17 所示，读者可以将继电器的触点引出，用来控制 220V 的电灯（虚线右边部分）。将 220V 市电由 AD 端输入，继电器控制电灯的亮灭。

注意：在实验中一定要注意安全!!!

图 9-17　单片机与继电器相连的控制电路

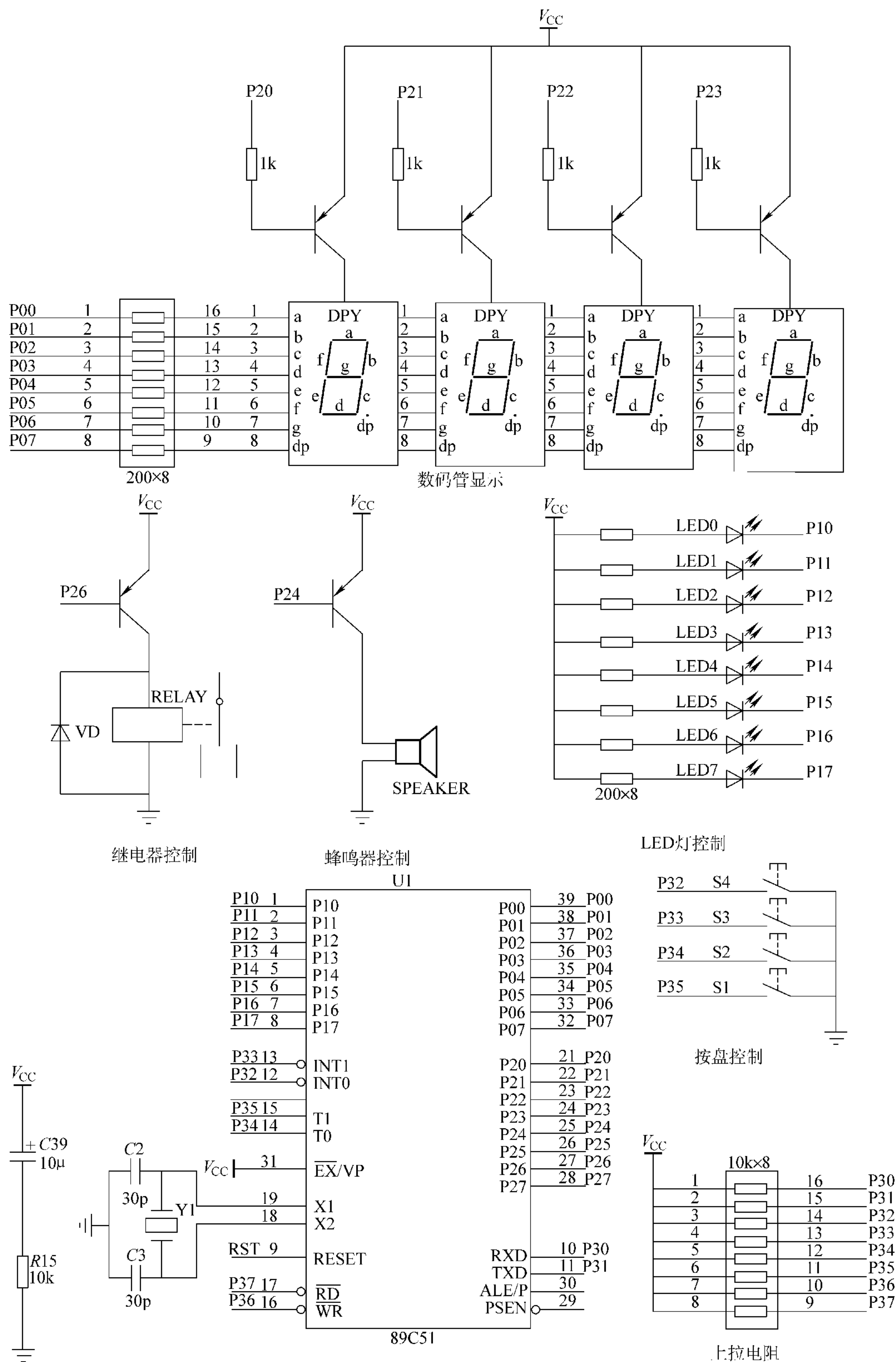
程序功能：利用单片机引脚 P2.6 输出高低电平，控制继电器吸合、释放，用继电器的常开触点控制电灯的亮灭。

程序代码：

```
#include <reg51.h>
#define uint unsigned int
sbit jdq = P2^6;                    // 定义 P2.6 口
void main(void)
{
uint i,j;
while(1)
    {
for(i = 1000;i > 0;i - -)
for(j = 1000;j > 0;j - -);         //延时
    jdq = ! jdq;                    // 对 P2.6 口取反,控制继电器吸合、释放
    }
}
```

9.8　单片机综合应用程序

本章讲述了基于单片机基础实例的应用，每一个应用实例都有很好的学习价值，在本节将把全章的基础实例综合在一起形成一个相对详细的综合实例。具体实例功能见表 9-5。



VCC
P20
P21
P22
P23
1k
1k
1k
1k
P00
P01
P02
P03
P04
P05
P06
P07
DPY
200×8
数码管显示
P26
RELAY
VD
继电器控制
P24
SPEAKER
蜂鸣器控制
LED0
LED1
LED2
LED3
LED4
LED5
LED6
LED7
P10
P11
P12
P13
P14
P15
P16
P17
LED灯控制
U1
89C51
EX/VP
X1
X2
RESET
INT1
INT0
T1
T0
RXD
TXD
ALE/P
PSEN
RD
WR
RST
C2
C3
30p
Y1
C39
10μ
R15
10k
P32 S4
P33 S3
P34 S2
P35 S1
按盘控制
10k×8
P30
P31
P36
P37
上拉电阻

图 9-18　硬件原理

表 9-5　综合实验功能

步骤	功能	备注
0（开机）	等待	数码管显示 0000
1（S1）	LED 灯闪动	数码管显示 0001
2（S2）	LED 灯流水动作	数码管显示 0002
3（S3）	蜂鸣器发声	数码管显示 0003
4（S4）	继电器吸合	数码管显示 0004

程序运行说明：系统开机默认为步骤 0，通过按键在各功能间进行切换。如按下 S1 则执行步骤 1，按下 S2 则执行步骤 2……

硬件原理如图 9-18 所示。

软件流程图如图 9-19 所示。

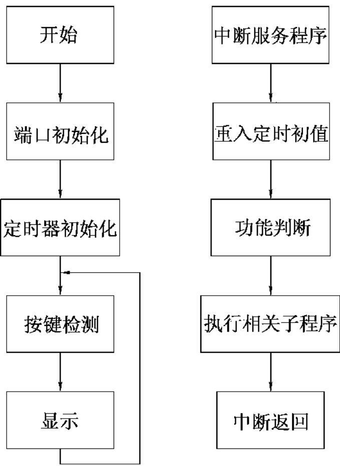


图 9-19　软件流程图

程序代码：

```
/******************************/
/*      杭州电子 & 计算机工作室      */
/*      http://www.hificat.com      */
/*      综合演示程序              */
/*      目标器件：AT89C51         */
/*      晶振：12MHz               */
/*      编译环境：Keil 7.50A      */
/******************************/
/*********************包含头文件*********************/
#include <reg51.h>
#include <absacc.h>
/********************共阳 LED 段码表********************/
unsigned char code tab[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80,
```

```
0x90};
/*******************定义全局变量******************/
int keyval;                                  //按键值
/*******************端口定义******************/
sfr key = 0xB0;                              //定义键盘口为 P3
sbit S4  = P3^5;                             //P35 定义为 S4
sbit S3 = P3^4;                              //P34 定义为 S3
sbit S2 = P3^3;                              //P33 定义为 S2
sbit S1 = P3^2;                              //P32 定义为 S1
sbit SPEAKER = P2^4;                         //蜂鸣器
sbit RELAY = P2^6;                           //继电器
/********************************************
函数功能：按键消抖延时程序
入口参数：
出口参数：
********************************************/
void delay (void)
{
    unsigned int i;
    for ( i =0; i < 300; i++ );
}
/********************************************
函数功能：LED 数码管显示程序
入口参数：k
出口参数：
********************************************/
void display (int k)
{
    P2 =0xfe;                                //位选
    P0 =tab [k/1000];                        //显示千位数字
    delay ();                                //延时
    P2 =0xfd;                                //位选
    P0 =tab [k%1000/100];                    //显示百位数字
    delay ();                                //延时
    P2 =0xfb;                                //位选
    P0 =tab [k%100/10];                      //显示十位数字
    delay ();                                //延时
    P2 =0xf7;                                //位选
    P0 =tab [k%10];                          //显示个位数字
```

```
    delay ( );                                  //延时
    P2 =0xff;                                   //位选
}
/***************************************
函数功能：LED 闪动延时程序
入口参数：
出口参数：
***************************************/
void leddelay (void)
{
    unsigned char i, j;
    for (i =0; i <255; i + +)
    for (j =0; j <255; j + +);
}
/***************************************
函数功能：LED 闪动程序
入口参数：
出口参数：
***************************************/
void ledflash (void)
{
    P1 =0xff;                                   //关所有 LED
    leddelay ( );                               //延时等待
    P1 =0x00;                                   //开所有 LED
    leddelay ( );
}
/***************************************
函数功能：LED 流水灯程序
入口参数：
出口参数：
***************************************/
void ledflow (void)
{
    unsigned char i;
    unsigned char temp;
    temp  =  0x01;                              //赋初值给 temp，只有 1 位为 1
        for (i  =  0; i  <  8; i + +)
         {
            P1  =  ~temp;                       //将 temp 取反后送 P1 口输出
```

```
            leddelay ( );                       //调用延时函数
            temp = temp << 1;                   //temp 中的数据左移 1 位
        }
}
/****************************************************
函数功能：蜂鸣器发声程序
入口参数：
出口参数：
****************************************************/
void sound (void)
{
    SPEAKER = 0;                                //蜂鸣器开
    delay ( );                                  //延时
    SPEAKER = 1;                                //蜂鸣器关
}
/****************************************************
函数功能：继电器吸合程序
入口参数：
出口参数：
****************************************************/
void relayon (void)
{
    RELAY = 0;                                  //继电器吸合
    leddelay ( );                               //延时
    RELAY = 1;                                  //继电器释放
}
/****************************************************
函数功能：主程序
入口参数：
出口参数：
****************************************************/
void main (void)
{
    keyval = 0;                                 //按键值清零
    key = 0xff;                                 //按键置输入状态
    P2 = 0xff;                                  //熄灭所有数码管
    P0 = 0xff;
/*******************定时器初始化*******************/
    EA = 1;                                     //中断总允许
```

```
    ET0 =1;                                     //T0 中断使能
    ET1 =1;                                     //T1 中断使能
    TMOD =0x11;                                 //定时器工作方式 1
    TH0 = -10000/256;                           //定时器 T0 的高四位赋值
    TL0 = -10000%256;                           //定时器 T0 的低四位赋值
    TH1 = -100/256;                             //定时器 T1 的高四位赋值
    TL1 = -100%256;                             //定时器 T1 的高四位赋值
    TR0 =1;                                     //开 T0 中断，启动定时器
    TR1 =1;                                     //开 T1 中断，启动定时器
while (1)                                       //主循环
     {
        display (keyval);                       //显示功能号
        if (keyval = =1)                        //是否是功能 1
        ledflash ();                            //调用 LED 闪烁程序
        if (keyval = =2)                        //是否是功能 2
        ledflow ();                             //调用 LED 流水灯程序
        if (keyval = =3)                        //是否是功能 3
        relayon ();                             //调用继电器程序
        if (keyval = =4)                        //是否是功能 4
        sound ();                               //调用蜂鸣器发声程序
        display (keyval);                       //显示功能号
    }
}
/*****************************************************
函数功能：中断服务程序 T0
入口参数：
出口参数：
*****************************************************/
void intserv1 (void) interrupt 1 using 1
{
    TR0 =0;                                     //关定时器 T0
    display (keyval);
    TH0 = -10000/256;                           //定时器 T0 的高四位赋值
    TL0 = -10000%256;                           //定时器 T0 的低四位赋值
    TR0 =1;                                     //启动定时器 T0
}
/*****************************************************
函数功能：中断服务程序 T1
入口参数：
```

出口参数：

```
****************************************/
void intserv3 (void) interrupt 3 using 3
{
    TR1 =0;                                //关定时器 T1
    if ((key&0x3c)! =0x3c)                 //判断是否有键按下
    delay ();                              //有键按下延时消抖
    if ((key&0x3c)! =0x3c)                 //再次检测按键，判断是否误操作
     {
        if (S1 = =0)                       //S1 按下
        keyval =1;
        if (S2 = =0)                       //S2 按下
        keyval =2;
        if (S3 = =0)                       //S3 按下
        keyval =3;
        if (S4 = =0)                       //S4 按下
        keyval =4;
    }
    TR1 =1;                                //启动定时器 T1
}
```

9.9 单片机串行口数据接收

在本书串行中断中介绍了串行口的原理与应用，本例将介绍用非中断方式单片机的串行口将 PC 端发送过来的信号接收并在数码管上显示出来。

硬件电路如图 9-20 所示，其中我们增加了数码管电路，用来显示从串行口接收到的数据。

1. 上位机软件设置

要在 PC 上向串口发送数据一定要借助相应软件。在调试过程中最常用的方法是利用系统自带的超级终端或用第三方软件——串口调试助手。串口调试助手在串行中断章节中也有介绍，它设置方便、灵活，界面简洁明了。本实例中使用串口调试助手向单片机发送一个数，使单片机收到后在数码管上显示出来。上位机端设置与串口调试助手界面如图 9-21 所示：

串口：COM1
波特率：9600
校验位：无
数据位：8 位
停止位：1 位
发送内容：0x55（十六进制）

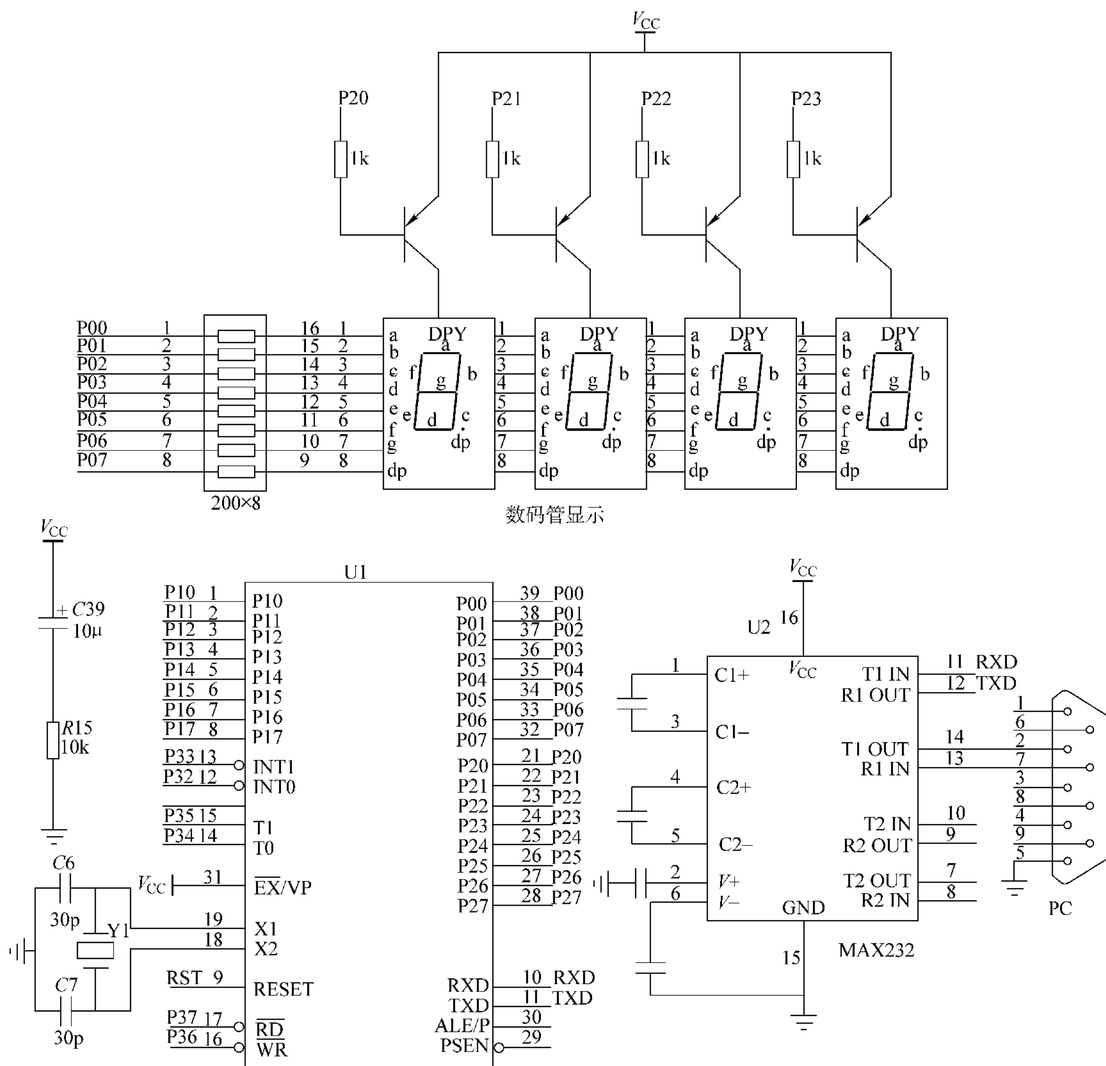


图 9-20　串行通信电路原理

2. 单片机程序

1）程序功能：用非中断方式将串口收到的数据在数码管上显示出来。

2）程序代码：

```
/****************************/
/*        杭州电子 & 计算机工作室          */
/*        http://www. hificat. com         */
/*        串行口演示程序                   */
/*        目标器件：AT89C51                */
/*        晶振：11. 0592MHz                */
/*        编译环境：Keil 7. 50A            */
/****************************/
/*********************包含头文件*********************/
```



图 9-21 串口调试软件界面

```
#include "reg51.h"
#include <absacc.h>
/******************共阳 LED 段码表*******************/
unsigned char code tab [ ] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80,
0x90};
/******************定义全局变量*********************/
unsigned char dat;
/********************************************
函数功能:串行口初始化程序
入口参数:
出口参数:
********************************************/
void Init_Com (void)
{
    TMOD = 0x20;                    //定时器工作方式 2, 初值自动装入
    PCON = 0x00;                    //波特率不增倍
    SCON = 0x50;                    //串行工作方式设定
    TH1 = 0xFd;                     //定时器初值高位
    TL1 = 0xFd;                     //定时器初值低位
    TR1 = 1;                        //启动定时器
```

```
}
/****************************************
函数功能：LED 数码管延时程序
入口参数：
出口参数：
****************************************/
void delay (void)
{
int k;
for (k =0; k <600; k + +);
}
/****************************************
函数功能：LED 数码管显示程序
入口参数：k
出口参数：
****************************************/
void display (int k)
{
    P2 =0xfe;                       //位选
    P0 =tab [k/1000];               //显示千位数字
    delay ();                       //延时
    P2 =0xfd;                       //位选
    P0 =tab [k%1000/100];           //显示百位数字
    delay ();                       //延时
    P2 =0xfb;                       //位选
    P0 =tab [k%100/10];             //显示十位数字
    delay ();                       //延时
    P2 =0xf7;                       //位选
    P0 =tab [k%10];                 //显示个位数字
    delay ();                       //延时
    P2 =0xff;                       //位选
}
/****************************************
函数功能：主程序
入口参数：
出口参数：
****************************************/
void main ()
{
```

```
    P2 =0xff;                              //端口初始化，关 LED 显示
    P0 =0xff;
    Init_Com ( ) ;                         //调用串口初始化程序
    while (1)                              //主循环
     {
        if ( RI )                          //判断是否收到数据
          {
            dat  =  SBUF;                  //接收数据
            RI  =  0;                      //软件清除标志位
          }
      display (dat) ;                      //显示收到的数据
    }
}
```

第 10 章　单片机高级应用实例

10.1　矩阵键盘应用实例

在前面的章节中介绍了独立按键的应用，独立按键具有编程简单、但占用 I/O 口资源的特点，不适合在按键较多的场合应用。在实际应用中经常要用到输入数字、字母等功能，如电子密码锁、电话机键盘等一般都至少有 12 ~ 16 个按键，在这种情况下如果用独立按键显然太浪费 I/O 口资源，为此我们引入了矩阵键盘的应用。

10.1.1　矩阵键盘简介

矩阵键盘又称行列键盘，它是用 4 条 I/O 线作为行线，4 条 I/O 线作为列线组成的键盘。在行线和列线的每个交叉点上设置一个按键。这样键盘上按键的个数就为 4 ×4 个。这种行列式键盘结构能有效地提高单片机系统中 I/O 口的利用率。

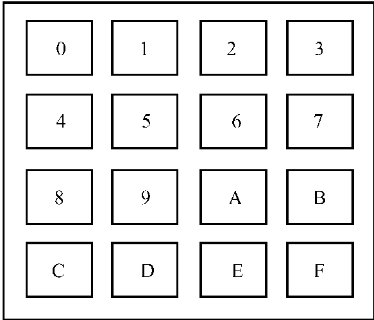


图 10-1　矩阵键盘布局

10.1.2　矩阵键盘的工作原理

最常见的键盘布局如图 10-1 所示。一般由 16 个按键组成，在单片机中正好可以用一个 P 口实现 16 个按键功能，这也是在单片机系统中最常用的形式，4 ×4 矩阵键盘的内部电路如图 10-2 所示。

当无按键闭合时，P10 ~ P13 与 P14 ~ P17 之间开路。当有按键闭合时，与闭合键相连的两条 I/O 口线之间短路。判断有无按键按下的方法是：第一步，置列线 P14 ~ P17 为输入状

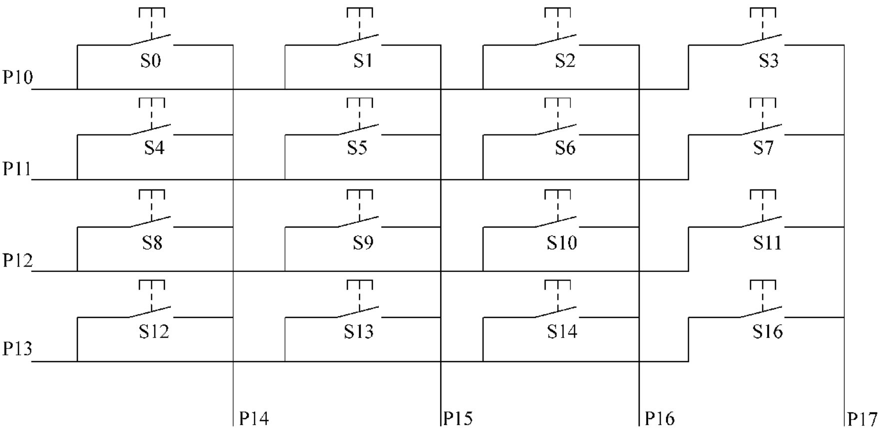


图 10-2　矩阵键盘内部电路

态，从行线 P10 ~ P13 输出低电平，读入列线数据，若某一列线为低电平，则该列线上有键闭合；第二步，行线轮流输出低电平，从列线 P14 ~ P17 读入数据，若有某一列为低电平，则对应行线上有键按下。综合一二两步的结果，可确定按键编号。但是键闭合一次只能进行一次键功能操作，因此须等到按键释放后，再进行键功能操作，否则按一次键，有可能会连续多次进行同样的键操作。

10.1.3 矩阵键盘软硬件设计实例

本小节利用配套开发板为硬件平台，介绍矩阵式键盘的编程方法。本实验通过按下相应键后在一位数码管上显示出键值。0 ~ 16 个键分别对应显示 0 ~ F。实验时，请使用 KC-103 单片机键盘板模块进行扩展，从而完成矩阵键盘的实验，如图 10-3 所示。

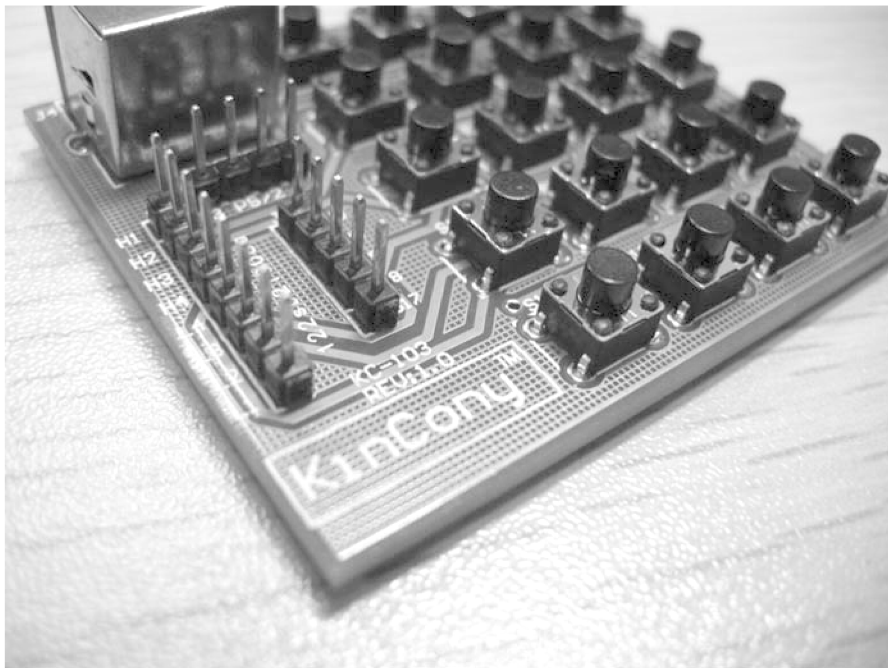


图 10-3　KC-103 单片机键盘板模块

1. 硬件原理图

本实验可以直接在配套开发板与 KC-103 单片机键盘板模块连接完成，其硬件原理如图 10-4 所示。

根据图 10-4 电路原理，键盘扫描方法是：行线 P10 ~ P13 为输出线，列线 P14 ~ P17 为输入线。一开始单片机将行线（P10 ~ P13）全部输出低电平，此时读入列线数据，若列线全为高电平则没有键按下，当列线有出现低电平时调用延时程序以此来去除按键抖动。延时完成后再判断是否有低电平，如果此时读入列线数据还是有低电平，则说明确实有键按下。最后一步确定键值。现在我们以第二行的 S5 键为例，若按下 S5 后我们应该怎么得到这个键值呢？当判断确实有键按下之后，行线轮流输出低电平，根据读入列线的数据可以确定键值。首先，单片机将 P10 输出为低电平，其他 P11 ~ P13 输出高电平，此时读取列线的数据全为高电平，说明没有在第一行有键按下；其次，单片机将 P11 输出低电平，其他 P10、

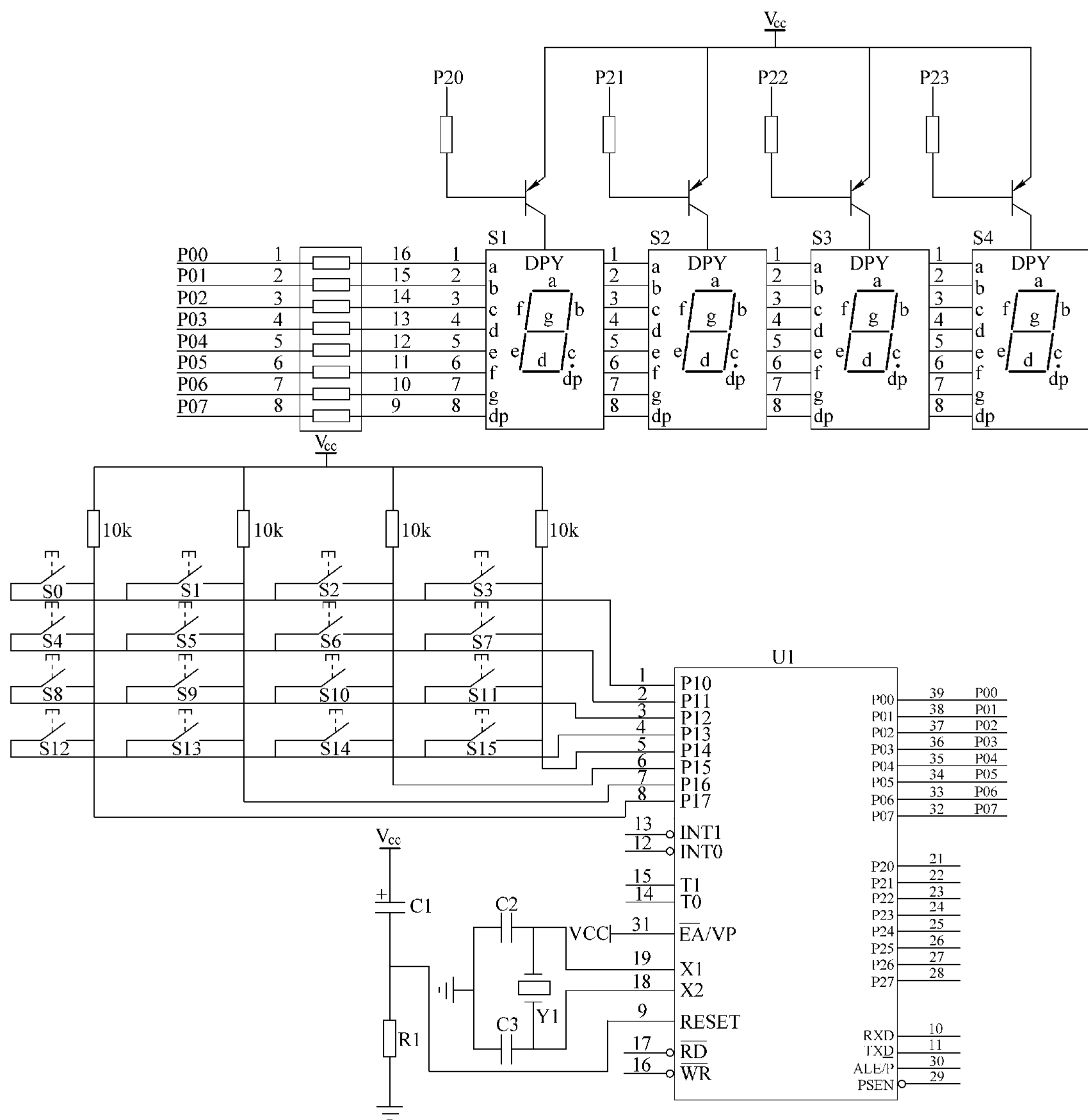


图 10-4　硬件原理图

P12、P13 仍为高电平，此时再来读取列线数据，发现列线读到的数据有低电平，数值为 1011（0x0B），如果我们的键盘布局已经确定，那么 0x0B 就代表 S5 的值了。转到 S5 键功能处理子程序就可以达到目的。

2. 程序流程图

程序流程如图 10-5 所示。

3. 软件代码

```
/*****************************/
/*杭州晶控电子有限公司                                    */
/*http://www.hificat.com                                 */
/*矩阵键盘测试程序                                        */
```

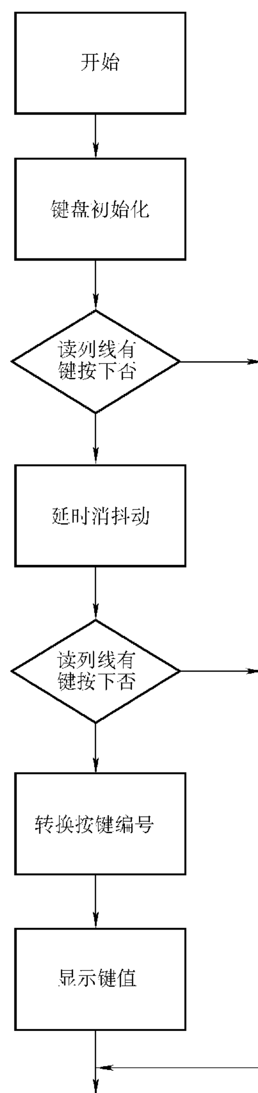


图 10-5　程序流程图

```
/*目标器件:AT89S51                                                    */
/*晶振:11.0592MHz                                                     */
/*编译环境:Keil 7.50A                                                  */
/***************************************************************************/
/************************包含头文件*********************/
#include <reg51.h>
/************************数码管表格*********************/
unsigned char table[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,
0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
/**************************************************************************
函数功能:延时子程序
```

```
入口参数:
出口参数:
*****************************************/
void delay(void)
{
    unsigned char i,j;
    for(i=0;i<20;i++)
    for(j=0;j<250;j++);
}
/*****************************************
函数功能:LED 显示子程序
入口参数:i
出口参数:
*****************************************/
void display(unsigned char i)
{
    P2=0xfe;
    P0=table[i];
}
/*****************************************
函数功能:键盘扫描子程序
入口参数:
出口参数:
*****************************************/
void keyscan(void)
{
    unsigned char n;
    //扫描第一行
    P1=0xfe;
    n=P1;
    n&=0xf0;
    if(n!=0xf0)
    {
        delay();
        P1=0xfe;
        n=P1;
        n&=0xf0;
        if(n!=0xf0)
        {
```

```
            switch(n)
            {
            case(0xe0):display(3);break;
            case(0xd0):display(2);break;
            case(0xb0):display(1);break;
            case(0x70):display(0);break;
            }
        }
}
//扫描第二行
P1 =0xfd;
n =P1;
n& =0xf0;
if(n!  =0xf0)
{
        delay();
        P1 =0xfd;
        n =P1;
        n& =0xf0;
        if(n!  =0xf0)
        {
            switch(n)
            {
            case(0xe0):display(7);break;
            case(0xd0):display(6);break;
            case(0xb0):display(5);break;
            case(0x70):display(4);break;
            }
        }
}
//扫描第三行
P1 =0xfb;
n =P1;
n& =0xf0;
if(n!  =0xf0)
{
        delay();
        P1 =0xfb;
        n =P1;
```

```
            n& = 0xf0;
            if(n!  = 0xf0)
            {
                switch(n)
                {
                case(0xe0):display(11);break;
                case(0xd0):display(10);break;
                case(0xb0):display(9);break;
                case(0x70):display(8);break;
                }
            }
        }
        //扫描第四行
        P1 = 0xf7;
        n = P1;
        n& = 0xf0;
        if(n!  = 0xf0)
        {
            delay();
            P1 = 0xf7;
            n = P1;
            n& = 0xf0;
            if(n!  = 0xf0)
            {
                switch(n)
                {
                case(0xe0):display(15);break;
                case(0xd0):display(14);break;
                case(0xb0):display(13);break;
                case(0x70):display(12);break;
                }
            }
        }
    }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能:主程序
入口参数:
出口参数:
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
```

```
void main(void)
{
    while(1)
    {
        keyscan();
    }
```

10.2 字符型 LCD 应用实例

在日常生活中，我们对液晶显示器并不陌生。液晶显示模块已作为很多电子产品的通用器件，如在计算器、万用表、电子表及很多家用电子产品中都可以看到，显示的主要是数字、专用符号和图形。在单片机的人机交流界面中，一般的输出方式有以下几种：发光管、LED 数码管、液晶显示器。发光管和 LED 数码管比较常用，软硬件都比较简单，在前面章节已经介绍过，在此不作介绍，本节重点介绍字符型液晶显示器的应用。

在单片机系统中应用晶液显示器作为输出器件有以下几个优点：

1）显示质量高：由于液晶显示器每一个点在收到信号后就一直保持那种色彩和亮度，恒定发光，而不像阴极射线管显示器（CRT）那样需要不断刷新新亮点。因此，液晶显示器画质高且不会闪烁。

2）数字式接口：液晶显示器都是数字式的，和单片机系统的接口更加简单可靠，操作更加方便。

3）体积小、重量轻：液晶显示器通过显示屏上的电极控制液晶分子状态来达到显示的目的，在重量上比相同显示面积的传统显示器要轻得多。

4）功耗低：相对而言，液晶显示器的功耗主要消耗在其内部的电极和驱动 IC 上，因而耗电量比其他显示器要少得多。

10.2.1 液晶显示概述

1. 液晶显示原理

液晶显示的原理是利用液晶的物理特性，通过电压对其显示区域进行控制，有电就有显示，这样即可以显示出图形。液晶显示器具有厚度薄、适用于大规模集成电路直接驱动、易于实现全彩色显示的特点，目前已经被广泛应用在便携式计算机、数字摄像机、PDA 移动通信工具等众多领域。

2. 液晶显示器的分类

液晶显示的分类方法有很多种，通常可按其显示方式分为段式、字符式、点阵式等。除了黑白显示外，液晶显示器还有多灰度彩色显示等。如果根据驱动方式来分，可以分为静态驱动（Static）、单纯矩阵驱动（Simple Matrix）和主动矩阵驱动（Active Matrix）三种。

3. 液晶显示器各种图形的显示原理

1）线段的显示：点阵图形式液晶由 $M \times N$ 个显示单元组成。假设 LCD 显示屏有 64 行，每行有 128 列，每 8 列对应 1 字节的 8 位，即每行由 16B，共 $16 \times 8 = 128$ 个点组成，屏上 64×16 个显示单元与显示 RAM 区 1024B 相对应，每一字节的内容和显示屏上相应位置的亮

暗对应。例如屏的第一行的亮暗由 RAM 区的 000H～00FH 的 16B 的内容决定，当（000H）＝FFH 时，则屏幕的左上角显示一条短亮线，长度为 8 个点；当（3FFH）＝FFH 时，则屏幕的右下角显示一条短亮线；当（000H）＝FFH，（001H）＝00H，（002H）＝00H，……（00EH）＝00H，（00FH）＝00H 时，则在屏幕的顶部显示一条由 8 段亮线和 8 条暗线组成的虚线。这就是 LCD 显示的基本原理。

2）字符的显示：用 LCD 显示一个字符时比较复杂，因为一个字符由 6×8 或 8×8 点阵组成，既要找到和显示屏幕上某几个位置对应的显示 RAM 区的 8B，还要使每字节的不同位为“1”，其他的为“0”，为“1”的点亮，为“0”的不亮。这样一来就组成某个字符。但对于内带字符发生器的控制器来说，显示字符就比较简单了，可以让控制器工作在文本方式，根据在 LCD 上开始显示的行列号及每行的列数找出显示 RAM 对应的地址，设立光标，在此送上该字符对应的代码即可。

3）汉字的显示：汉字的显示一般采用图形的方式，事先从微机中提取要显示的汉字的点阵码（一般用字模提取软件），每个汉字占 32B，分左右两半，各占 16B，左边为 1、3、5…右边为 2、4、6…，根据在 LCD 上开始显示的行列号及每行的列数可找出显示 RAM 对应的地址，设立光标，送上要显示的汉字的第一字节，光标位置加 1，送第二个字节，换行按列对齐，送第三个字节……，直到 32B 显示完就可以在 LCD 上得到一个完整汉字。

10.2.2 1602 字符型 LCD 简介

字符型液晶显示模块是一种专门用于显示字母、数字、符号等点阵式 LCD，目前常用 16×1，16×2，20×2 和 40×2 行等的模块。下面以长沙太阳人电子有限公司的 1602 字符型液晶显示器为例，介绍其用法。一般 1602 字符型液晶显示器实物如图 10-6、图 10-7 所示。外形尺寸如图 10-8 所示。



图 10-6 1602 字符型液晶屏正面

1. 1602LCD 主要技术参数

显示容量为 16×2 个字符；

芯片工作电压为 4.5～5.5V；

工作电流为 2.0mA（5.0V）；

模块最佳工作电压为 5.0V；

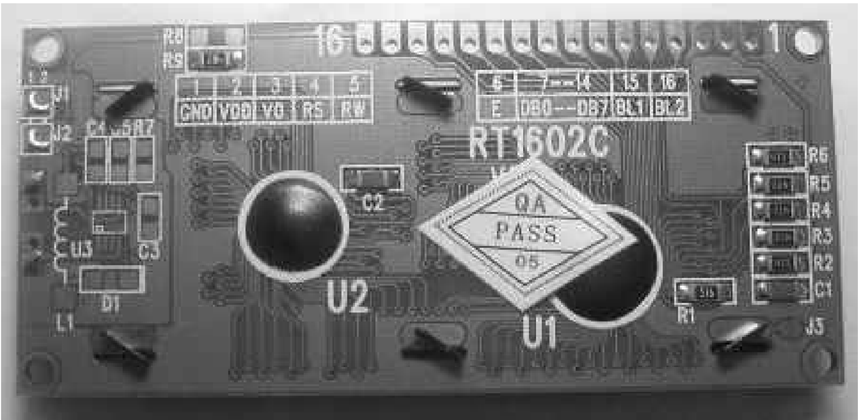


图 10-7　1602 字符型液晶屏反面

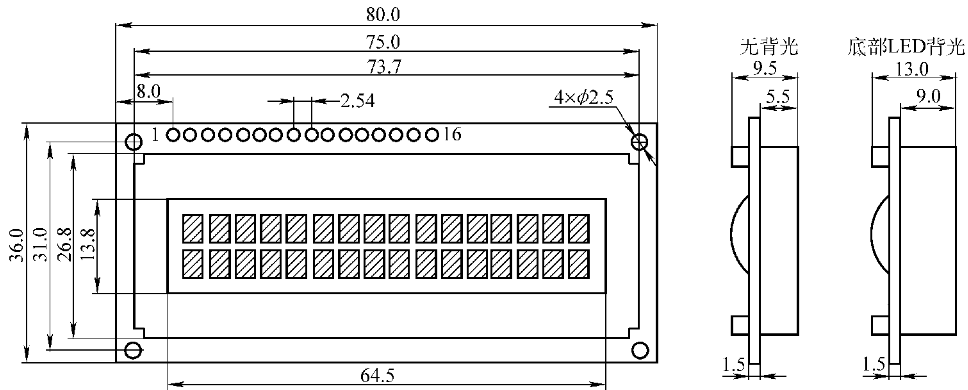


图 10-8　1602 字符型液晶屏尺寸标准

字符尺寸 $W \times H$ 为 2.95 ×4.35mm。

2. 接口、信号说明

1602LCD 采用标准的 14 引脚（无背光）或 16 引脚（带背光）接口，各引脚接口说明见表 10-1。

表 10-1　1602 字符型液晶屏接口引脚定义

引脚号	符号	引脚说明	引脚号	符号	引脚说明
1	V_{SS}	电源地	9	D2	数据
2	V_{DD}	电源正极	10	D3	数据
3	V_L	液晶显示偏压	11	D4	数据
4	RS	数据/命令选择	12	D5	数据
5	R/W	读/写选择	13	D6	数据
6	E	使能信号	14	D7	数据
7	D0	数据	15	BLA	背光源正极
8	D1	数据	16	BLK	背光源负极

引脚 1：V_{SS}为地电源。

引脚 2：V_{DD}接 5V 正电源。

引脚 3：V_L 为液晶显示器对比度调整端。接正电源时对比度最低，接地时对比度最高。对比度过高时会产生“鬼影”，使用时可以通过一个 10kΩ 的电位器调整对比度。

引脚 4：RS 为寄存器选择。高电平时选择数据寄存器、低电平时选择指令寄存器。

引脚 5：R/W 为读写信号线。高电平时进行读操作，低电平时进行写操作。当 RS 和 R/W 同为低电平时可以写入指令或者显示地址。当 RS 为低电平 R/W 为高电平时可以读忙信号，当 RS 为高电平 R/W 为低电平时可以写入数据。

引脚 6：E 为使能端，当 E 由高电平跳变成低电平时，液晶模块执行命令。

引脚 7 ~ 14：D0 ~ D7 为 8 位双向数据线。

引脚 15：背光源正极。

引脚 16：背光源负极。

3. 控制器接口说明（以 HD44780 及兼容芯片为例）

1）基本操作时序见表 10-2。

表 10-2　基本操作时序

读状态	输入	RS = L，R/W = H，E = H	输出	D0 ~ D7 = 状态字
写指令	输入	RS = L，R/W = L，D0 ~ D7 = 指令码，E = 高脉冲	输出	无
读数据	输入	RS = H，R/W = H，E = H	输出	D0 ~ D7 = 数据
写数据	输入	RS = H，R/W = L，D0 ~ D7 = 数据，E = 高脉冲	输出	无

读写操作时序分别如图 10-9、图 10-10 所示。

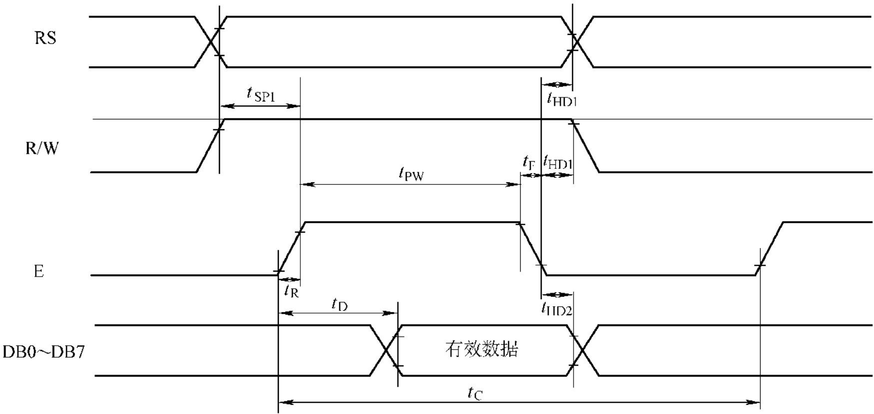


图 10-9　读操作时序

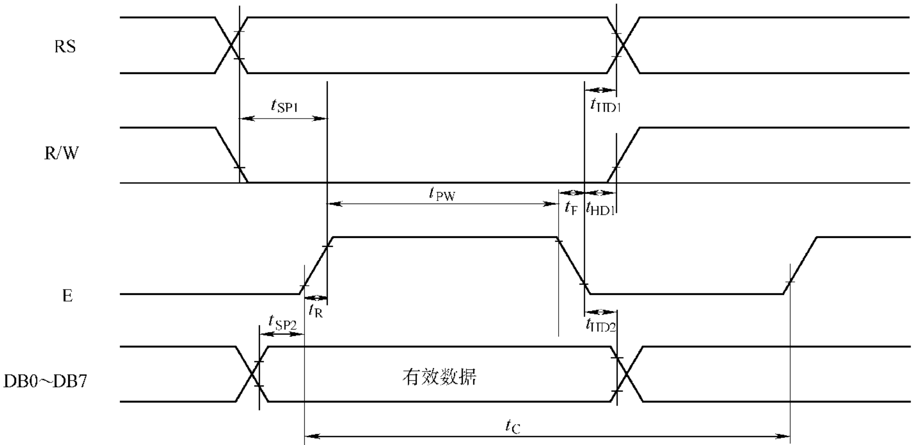


图 10-10　写操作时序

2）RAM 地址映射：液晶显示模块是一个慢显示器件，所以在执行每条指令之前一定要确认模块的忙标志为低电平（表示不忙），否则此指令失效。要显示字符时要先输入显示字符地址，也就是告诉模块在哪里显示字符，1602 字符型液晶屏的内部显示地址如图 10-11 所示。

例如第二行第一个字符的地址是 40H，那么是否直接写入 40H 就可以将光标定位在第二行第一个字符的位置呢？这样不行，因为写入显示地址时要求最高位 D7 恒定为高电平 1，所以实际写入的数据应该是 01000000B（40H）+10000000B（80H）=11000000B（C0H）。

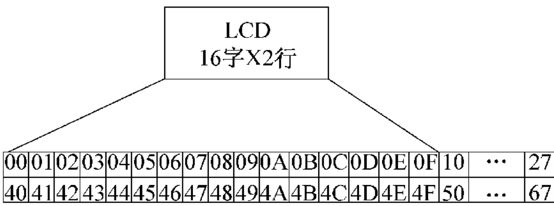


图 10-11　RAM 地址映射

程序在开始时对液晶模块功能进行了初始化设置，约定了显示格式。注意显示字符时光标是自动右移的，无需人工干预，每次输入指令都先调用判断液晶模块是否忙的子程序 DELAY，然后输入显示位置的地址 0C0H，最后输入要显示的字符 A 的代码 41H。

4. 指令说明

1602 液晶模块内部的控制器共有 11 条控制指令，见表 10-3。

表 10-3　1602 液晶屏指令说明

序号	指令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	置功能	0	0	0	0	1	DL	N	F	*	*
7	置字符发生存储器地址	0	0	0	1	字符发生存储器地址					
8	置数据存储器地址	0	0	1	显示数据存储器地址						
9	读忙标志或地址	0	1	BF	计数器地址						
10	写数到 CGRAM 或 DDRAM	1	0	要写的数据内容							
11	从 CGRAM 或 DDRAM 读数	1	1	读出的数据内容							

它的读写操作、屏幕和光标的操作都是通过指令编程来实现的（说明：1 为高电平、0 为低电平）。

指令 1：清显示，指令码 01H，光标复位到地址 00H 位置。

指令 2：光标复位，光标返回到地址 00H。

指令 3：光标和显示模式设置。I/D：光标移动方向，高电平右移，低电平左移。S：屏幕上所有文字是否左移或者右移。高电平表示有效，低电平则无效。

指令 4：显示开关控制。D：控制整体显示的开与关，高电平表示开显示，低电平表示关显示。C：控制光标的开与关，高电平表示有光标，低电平表示无光标。B：控制光标是否闪烁，高电平闪烁，低电平不闪烁。

指令 5：光标或显示移位。S/C：高电平时移动显示的文字，低电平时移动光标。

指令 6：功能设置命令。DL：高电平时为 4 位总线，低电平时为 8 位总线。N：低电平时为单行显示，高电平时双行显示。F：低电平时显示 5x7 的点阵字符，高电平时显示 5x10 的点阵字符。

指令 7：字符发生器 RAM 地址设置。

指令 8：DDRAM 地址设置。

指令 9：读忙信号和光标地址。BF：为忙标志位，高电平表示忙，此时模块不能接收命令或者数据，如果为低电平表示不忙。

指令 10：写数据。

指令 11：读数据。

5. 标准字库表

1602 液晶模块内部的字符发生存储器（CGROM）已经存储了 160 个不同的点阵字符图形，如图 10-12 所示。这些字符有：阿拉伯数字、英文字母的大小写、常用的符号和日文假名等，每一个字符都有一个固定的代码，比如大写的英文字母“A”的代码是 01000001B（41H），显示时模块把地址 41H 中的点阵字符图形显示出来，我们就能看到字母“A”

高位 / 低位	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
××××0000	CGRAM (1)		0	ə	P	\	p		—	タ	ミ	α	P
××××0001	(2)	!	1	A	Q	a	q	ロ	ア	チ	ム	ä	q
××××0010	(3)	″	2	B	R	b	r	r	イ	川	メ	β	θ
××××0011	(4)	#	3	C	S	c	s	」	ウ	ラ	モ	ε	∞
××××0100	(5)	$	4	D	T	d	t	\	エ	ト	セ	μ	Ω
××××0101	(6)	%	5	E	U	e	u	ロ	オ	ナ	ユ	B	0
××××0110	(7)	&	6	F	V	f	v	テ	カ	ニ	ヨ	P	Σ
××××0111	(8)	>	7	G	W	g	w	ア	キ	ヌ	ラ	g	π
××××1000	(1)	(	8	H	X	h	x	イ	ク	ネ	リ	∫	X
××××1001	(2)	)	9	I	Y	i	y	ウ	ケ	j	ル	-1	y
××××1010	(3)	#	:	J	Z	j	z	エ	コ	リ	レ	j	千
××××1011	(4)	+	:	K	[	k	{	オ	サ	ヒ	ロ	x	万
××××1100	(5)	フ	<	L	¥	l	|	セ	シ	フ	ワ	¢	[illegible]
××××1101	(6)	—	=	M	]	m	}	ユ	ス	ヘ	ソ	キ	+
××××1110	(7)	.	>	N	^	n	-	ヨ	セ	ホ	ハ	$\bar{n}$	
××××1111	(8)	/	?	O	—	o	←	ツ	ソ	マ	ロ	ö	

图 10-12　字符代码与字符图形的对应关系

6. 一般初始化（复位）过程

延时 15ms

写指令 38H（不检测忙信号）

延时 5ms

写指令 38H（不检测忙信号）

延时 5ms

写指令 38H（不检测忙信号）

以后每次写指令、读/写数据操作均需要检测忙信号

写指令 38H：显示模式设置

写指令 08H：显示关闭

写指令 01H：显示清屏

写指令 06H：显示光标移动设置

写指令 0CH：显示开及光标设置

7. 1602LCD 应用

1）硬件原理：1602 液晶显示模块可以和单片机 AT89C51 直接接口，电路原理如图 10-13 所示。

图 10-13　1602 液晶屏与 51 单片机的接口电路

2）软件功能：在 1602LCD 第一行显示网站名：www. hificat. com 在第二行显示联系电话：0571－85956028。软件流程图如图 10-14 所示。

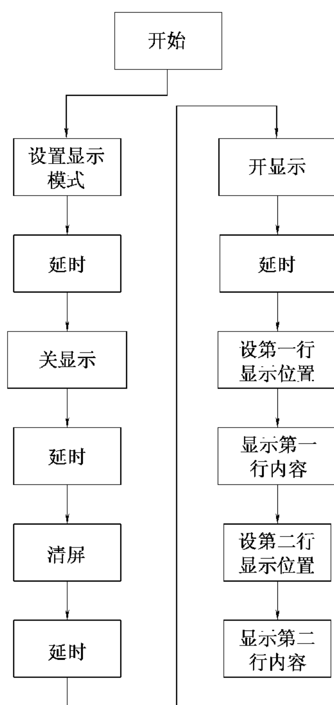


图 10-14　软件流程图

3）程序代码

```
/*********************************/
/*        杭州晶控电子有限公司计算机工作室        */
/*        http://www.hificat.com                  */
/*        1602LCD 演示程序                         */
/*        目标器件:AT89C51                         */
/*        晶振:12MHz                               */
/*        编译环境:Keil 7.50A                      */
/*********************************/
#include <reg51.h>
#include <intrins.h>

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef bit BOOL ;

sbit rs  = P3^5;      //
sbit rw  = P3^6;
sbit ep  = P3^7;
```

```
BYTE code dis1[] = {"www.hificat.com"};
BYTE code dis2[] = {"0571-85956028"};

/**************************************************
函数功能:延时子程序
入口参数:ms
出口参数:
**************************************************/
void delay(BYTE ms)
{                              // 延时子程序
        BYTE i;
        while(ms--)
        {
             for(i = 0; i< 250; i++)
             {
                  _nop_();
                  _nop_();
                  _nop_();
                  _nop_();
             }
        }
}

/**************************************************
函数功能:测试 LCD 忙碌状态
入口参数:
出口参数:result
**************************************************/
BOOL lcd_bz()
{
     BOOL result;
     rs = 0;
     rw = 1;
     ep = 1;
     _nop_();
     _nop_();
     _nop_();
     _nop_();
```

```
    result = (BOOL)(P1 & 0x80);
    ep = 0;
    return result;
}

/*****************************************************
函数功能:写指令数据到 LCD
入口参数:cmd
出口参数:
*****************************************************/
lcd_wcmd(BYTE cmd)
{                         // 写入指令数据到 LCD
    while(lcd_bz());
    rs = 0;
    rw = 0;
    ep = 0;
    _nop_();
    _nop_();
    P1 = cmd;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    ep = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    ep = 0;
}

/*****************************************************
函数功能:设定显示位置
入口参数:pos
出口参数:
*****************************************************/
lcd_pos(BYTE pos)
{                         //设定显示位置
```

```
    lcd_wcmd(pos | 0x80);
}

/*****************************************************
函数功能:写入显示数据到 LCD
入口参数:dat
出口参数:
*****************************************************/
lcd_wdat(BYTE dat)
{                          //写入字符显示数据到 LCD
    while(lcd_bz());
    rs = 1;
    rw = 0;
    ep = 0;
    P1 = dat;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    ep = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    ep = 0;
}

/*****************************************************
函数功能:LCD 初始化
入口参数:
出口参数:
*****************************************************/
lcd_init()
{                          //LCD 初始化设定
        lcd_wcmd(0x38);//
        delay(1);
        lcd_wcmd(0x0c);//
        delay(1);
```

```
        lcd_wcmd(0x06); //
        delay(1);
        lcd_wcmd(0x01); //清除 LCD 的显示内容
        delay(1);
}

/********************************************
函数功能:主程序
入口参数:
出口参数:
********************************************/
main()
{
        BYTE i;
        lcd_init();          // 初始化 LCD
        delay(10);
        lcd_pos(0);          // 设置显示位置为第一行的第 5 个字符
        i = 0;
        while(dis1[i] ! = '\0')
        {                    // 显示字符 hificat. com
            lcd_wdat(dis1[i]);
            i++;
        }
        lcd_pos(0x41);       // 设置显示位置为第二行第二个字符
        i = 0;
        while(dis2[i] ! = '\0')
        {
            lcd_wdat(dis2[i]); // 显示字符"0571 - 85956028
            i++;
        }
        while(1); //
}
```

10.3 步进电动机应用实例

步进电动机作为执行元件，是机电一体化的关键产品之一。步进电动机广泛应用于对精度要求比较高的运动控制系统中，如机器人、打印机、软盘驱动器、绘图仪、机械阀门控制

器等。随着微电子和计算机技术的发展，步进电动机的需求量与日俱增，在各个国民经济领域都有应用。本节重点介绍步进电动机的原理及应用。

10.3.1 步进电动机概述

1. 步进电动机原理

步进电动机是将电脉冲信号转变为角位移或线位移的开环控制元件。在非超载的情况下，电动机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响。当步进驱动器接收到一个脉冲信号，它就驱动步进电动机按设定的方向转动一个固定的角度（称为“步距角”），它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电动机转动的速度和加速度，从而达到调速的目的。步进电动机可以作为一种控制用的特种电动机，利用其没有积累误差（精度为100%）的特点，广泛应用于各种开环控制。常用的步进电动机实物如图10-15所示。步进电动机的内部结构如图10-16所示。

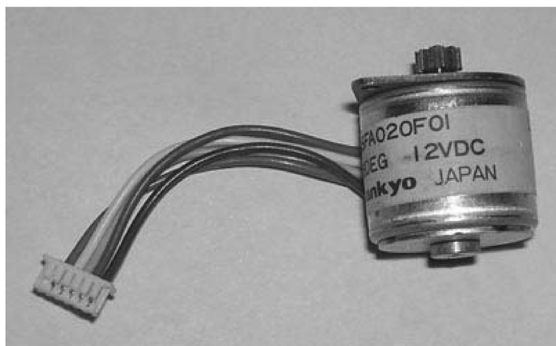


图10-15 步进电动机实物图

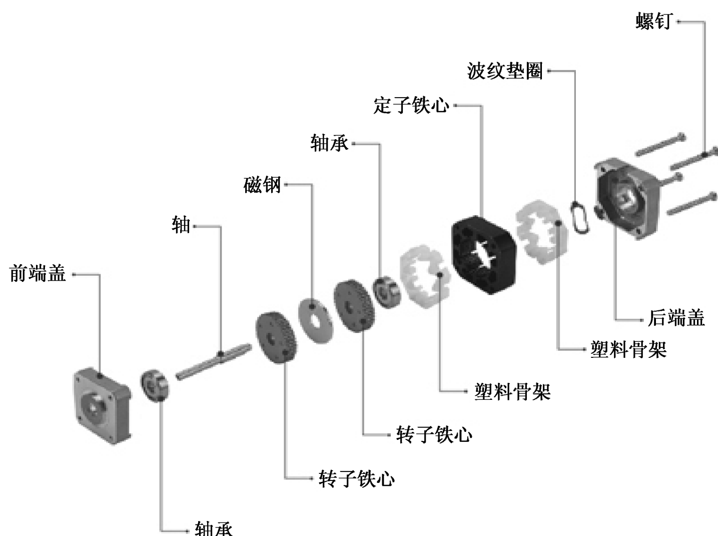


图10-16 步进电动机内部结构

2. 步进电动机分类

现在比较常用的步进电动机分为3种：反应式步进电动机（VR）、永磁式步进电动机

（PM）、混合式步进电动机（HB）。本节以反应式步进电动机为例，介绍其基本原理与应用方法。反应式步进电动机可实现大转矩输出，步进角一般为 1.5°。反应式步进电动机的转子磁路由软磁材料制成，定子上有多相励磁绕组，利用磁导的变化产生转矩。

常用步进电动机的内部模型如图 10-17 ~ 图 10-20 所示。

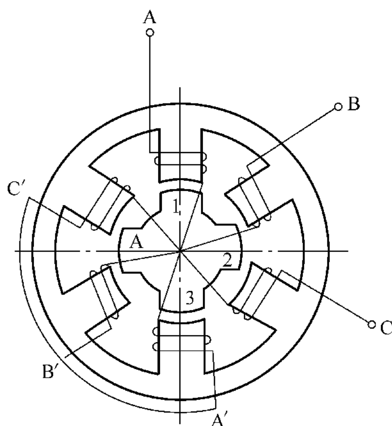


图 10-17　三相反应式步进电动机模型

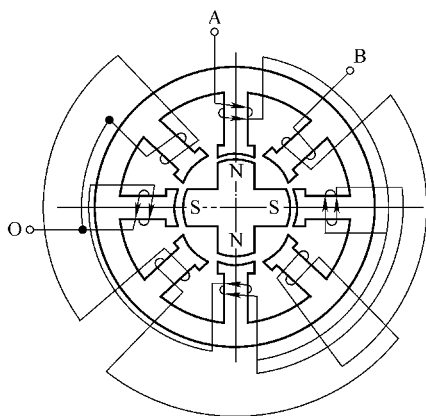


图 10-18　永磁式步进电动机

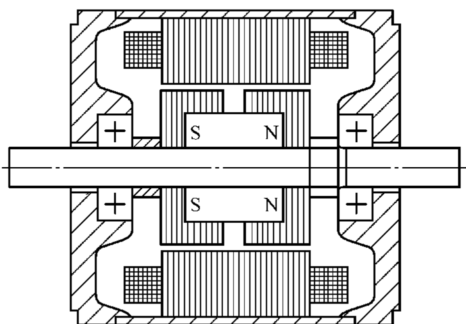


图 10-19　混合式步进电动机

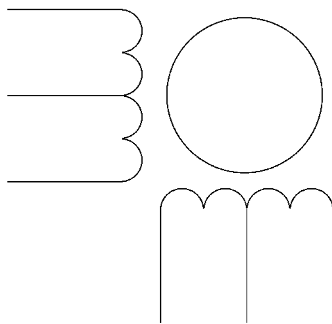


图 10-20　步进电动机的绕组

3. 步进电动机结构

步进电动机转子均匀分布着很多小齿，定子齿有 3 个励磁绕组，其几何轴线依次分别与转子齿轴线错开 0、1/3τ、2/3τ，（相邻两转子齿轴线间的距离为齿距，用τ表示），即 A 与齿 1 相对齐，B 与齿 2 向右错开 1/3τ，C 与齿 3 向右错开 2/3τ，A’与齿 5 相对齐（A’就是 A，齿 5 就是齿 1）。定、转子的展开图如图 10-21 所示。

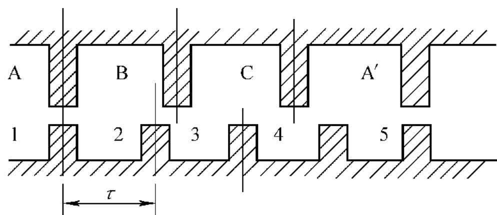


图 10-21　定、转子展开图

4. 步进电动机的旋转

如 A 相通电，B，C 相不通电时，由于磁场作用，齿 1 与 A 对齐（转子不受任何力，以

下均同)。

如B相通电，A，C相不通电时，齿2应与B对齐，此时转子向右移过1/3て，此时齿3与C偏移为1/3て，齿4与A’偏移2/3て。

如C相通电，A，B相不通电，齿3应与C对齐，此时转子又向右移过1/3て，此时齿4与A’偏移为1/3て。

如A相通电，B，C相不通电，齿4与A’对齐，转子又向右移过1/3て这样经过A、B、C、A分别通电状态，齿4（即齿1前一齿）移到A相，电动机转子向右转过一个齿距，如果不断地按A，B，C，A……通电，电动机就每步（每脉冲）1/3て地向右旋转。如按A，C，B，A……通电，电动机就反转。

由此可见：电动机的位置和速度与导电次数（脉冲数）和频率成一一对应关系。而方向由导电顺序决定。

不过，出于对转矩、平稳、噪声及减少角度等方面考虑。往往采用A-AB-B-BC-C-CA-A这种导电状态，这样将原来每步1/3て改变为1/6て。甚至于通过二相电流不同的组合，使其1/3て变为1/12て，1/24て，这就是电动机细分驱动的基本理论依据。

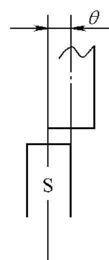


图10-22 磁通量

5. 步进电动机的转矩：

电动机一旦通电，在定、转子间将产生磁场（磁通量 Φ），当转子与定子错开一定角度产生力 F 与（$\mathrm{d}\Phi/\mathrm{d}\theta$）成正比。如图10-22所示。

其磁通量

$$\Phi = BS$$

式中，B 为磁通密度；S 为导磁面积。

F 与 LDB 成正比

其中，L 为铁心有效长度；D 为转子直径。

$$B = NI/R$$

式中，NI 为励磁绕组安匝数（电流乘匝数）；R 为磁阻。

$$转矩 = 力 \times 半径$$

转矩与电动机有效体积×安匝数×磁密成正比（只考虑线性状态）。因此，电动机有效体积越大，励磁安匝数越大，定转子间气隙越小，电动机转矩越大，反之亦然。

10.3.2 步进电动机的基本参数

1. 电动机固有步距角

电动机固有步距角表示控制系统每发一个步进脉冲信号，电动机所转动的角度。一般电动机出厂时都给出了一个步距角的值，如86BYG250A型电动机给出的值为0.9°/1.8°（表示半步工作时为0.9°、整步工作时为1.8°），这个步距角可以称之为‘电动机固有步距角’，它不一定是电动机实际工作时的真正步距角，真正的步距角还和驱动器有关。电动机转子转过的角位移用 θ 表示。$\theta = 360°/$（转子齿数 $J\times$ 运行拍数）。以常规二、四相，转子齿为50齿电动机为例，四拍运行时步距角为 $\theta = 360°/(50\times4) = 1.8°$（俗称整步）；八拍运行时步距角为 $\theta = 360°/(50\times8) = 0.9°$（俗称半步）。

2. 步进电动机的相数

步进电动机的相数是指电动机内部产生不同对极N、S磁场的励磁绕组对数，常用 m 表

示。目前常用的有二相、三相、四相、五相步进电动机。电动机相数不同，其步距角也不同，一般二相电动机的步距角为 0.9°/1.8°、三相的为 0.75°/1.5°、五相的为 0.36°/0.72° 。在没有细分驱动器时，用户主要靠选择不同相数的步进电动机来满足自己步距角的要求。

3. 保持转矩（HOLDING TORQUE）

保持转矩是指步进电动机通电但没有转动时，定子锁住转子的力矩。它是步进电动机最重要的参数之一，通常步进电动机在低速时的转矩接近保持转矩。由于步进电动机的输出转矩随速度的增大而不断衰减，输出功率也随速度的增大而变化，所以保持转矩就成为了衡量步进电动机最重要的参数之一。比如，当人们说 2Nm 的步进电动机，在没有特殊说明的情况下是指保持转矩为 2Nm 的步进电动机。

4. 定位转矩

定位转矩（DETENT TORQUE）是指步进电动机没有通电的情况下，电动机转子自身的锁定转矩。由于反应式步进电动机的转子不是永磁材料，所以它没有定位转矩。

5. 静转矩

静转矩表示电动机在额定静态电作用下，电动机不作旋转运动时，电动机转轴的锁定转矩。此转矩是衡量电动机体积（几何尺寸）的标准，与驱动电压及驱动电源等无关。

6. 拍数

步进电动机拍数是指完成一个磁场周期性变化所需脉冲数或导电状态，用 n 表示，或指电动机转过一个齿距角所需脉冲数。以四相电动机为例，有四相四拍运行方式即 AB-BC-CD-DA-AB，四相八拍运行方式即 A-AB-B-BC-C-CD-D-DA-A。

除了以上常用的参数外，步进电动机还有很多其他参数，如步距角精度、失步、失调角、最大空载起动频率等，可以查阅相关技术资料，在此不一一介绍。

10.3.3 步进电动机的驱动

目前，对步进电动机的控制主要有由分立器件组成的环形脉冲分配器、软件环形脉冲分配器、专用集成芯片环形脉冲分配器等，各种驱动器各有其优缺点。

1. 步进电动机的励磁方式

步进电动机的励磁方式一般分为 1 相励磁、2 相励磁、1～2 相励磁。其中 1 相励磁最为简单，转矩最小；2 相励磁可有较大的转矩；1～2 相励磁是属于半步的方式，也就是说旋转角度为前两种方式的一半。

3 种励磁方式分别见表 10-4、表 10-5、表 10-6。

表 10-4　1 相励磁

步数	A	B	/A	/B	步数	A	B	/A	/B
1	1	0	0	0	5	1	0	0	0
2	0	1	0	0	6	0	1	0	0
3	0	0	1	0	7	0	0	1	0
4	0	0	0	1	8	0	0	0	1

表 10-5　2 相励磁

步数	A	B	/A	/B	步数	A	B	/A	/B
1	1	1	0	0	5	1	1	0	0
2	0	1	1	0	6	0	1	1	0
3	0	0	1	1	7	0	0	1	1
4	1	0	0	1	8	1	0	0	1

表 10-6　1～2 相励磁

步数	A	B	/A	/B	步数	A	B	/A	/B
1	1	0	0	0	5	0	0	1	0
2	1	1	0	0	6	0	0	1	1
3	0	1	0	0	7	0	0	0	1
4	0	1	1	0	8	1	0	0	1

2. 步进电动机正反转控制

在步进电动机转动过程中改变绕组励磁顺序即可改变转动方向。

3. 步进电动机控制系统框图

一般一个完整的步进电动机控制系统包括控制器、驱动器、电动机三部分。其框图如图 10-23 所示。

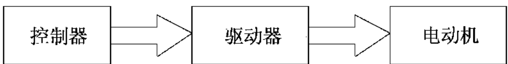


图 10-23　步进电动机控制系统框图

4. 小型步进电动机现场应用驱动电路

在本节中使用的是小型步进电动机，对电压和电流要求不是很高，为了说明应用原理，故采用最简单的驱动电路，目的在于验证步进电动机的使用，在正式工业控制中还需在此基础上改进。一般的驱动电路如图 10-24 所示。

在实际应用中一般驱动路数不止一路，用图 10-24 所示的分立器件组成的电路体积大，很多场合用现成的集成电路作为多路驱动。常用的小型步进电动机驱动电路可以用 ULN2003 或 ULN2803。本书配套实验板上用的是 ULN2003。ULN2003 是高压大电流达林顿晶体管阵列系列产品，具有电流增益高、工作电压高、温度范围宽、带负载能力强等特点，适应于各类要求高速大功率驱动的系统。ULN2003A 由 7 组达林顿晶体管阵列和相应的电阻网络以及钳位二极管网络构成，具有同时驱动 7 组负载的能力，为单片双极型大功率高速集成电路。ULN2003 内部结构如图 10-25 所示。ULN2003A 型高压大电流达林顿晶体管阵列电路的典型应用电路的等效电路如图 10-26 所示。钳位二极管用于保护绕组通断时的反电动势击穿集成电路，可以看出，该电路的应用非常简单。ULN2003A 的应用电路及本书配套实验板的原理分别如图 10-27、图 10-28 所示。

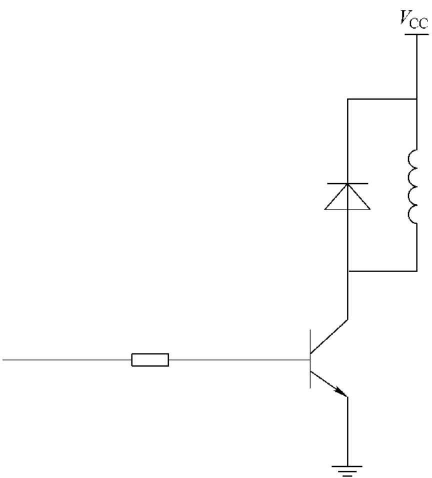


图 10-24　小型步进电动机驱动电路

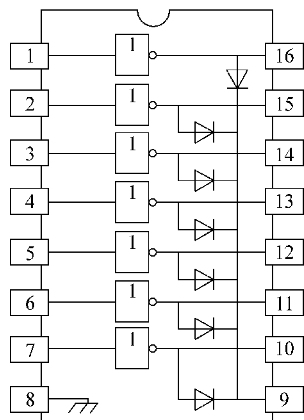


图 10-25　ULN2003 内部结构

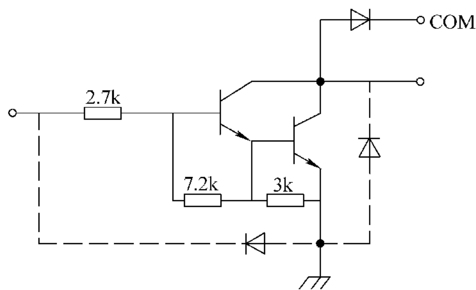


图 10-26　单路等效图

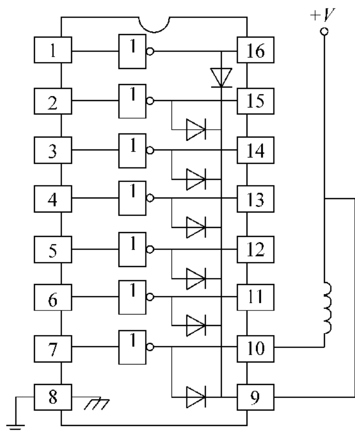


图 10-27　应用图

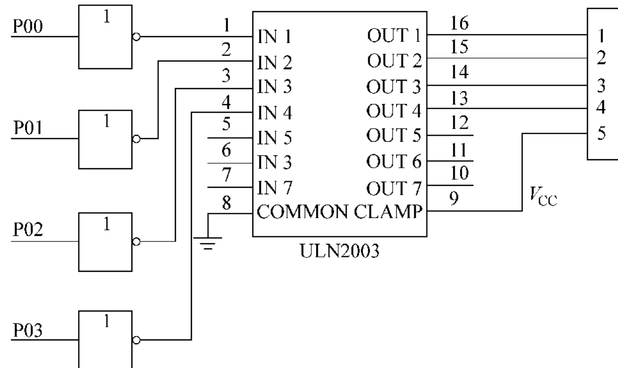


图 10-28　实验板上配套原理

5. 步进电动机的软硬件设计

1）实现功能：开机电动机正转，按住按键 S1 时反转。硬件原理如图 10-29 所示。

2）软件设计：电动机正反转的环形脉冲分配分别见表 10-7 和表 10-8。软件流程如图 10-30 所示。

表 10-7　正转环形脉冲分配表

步数	P00	P01	P02	P03
	A	B	/A	/B
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

表 10-8　反转环形脉冲分配表

步数	P00	P01	P02	P03
	A	B	/A	/B
1	1	1	0	0
2	1	0	0	1
3	0	0	1	1
4	0	1	1	0

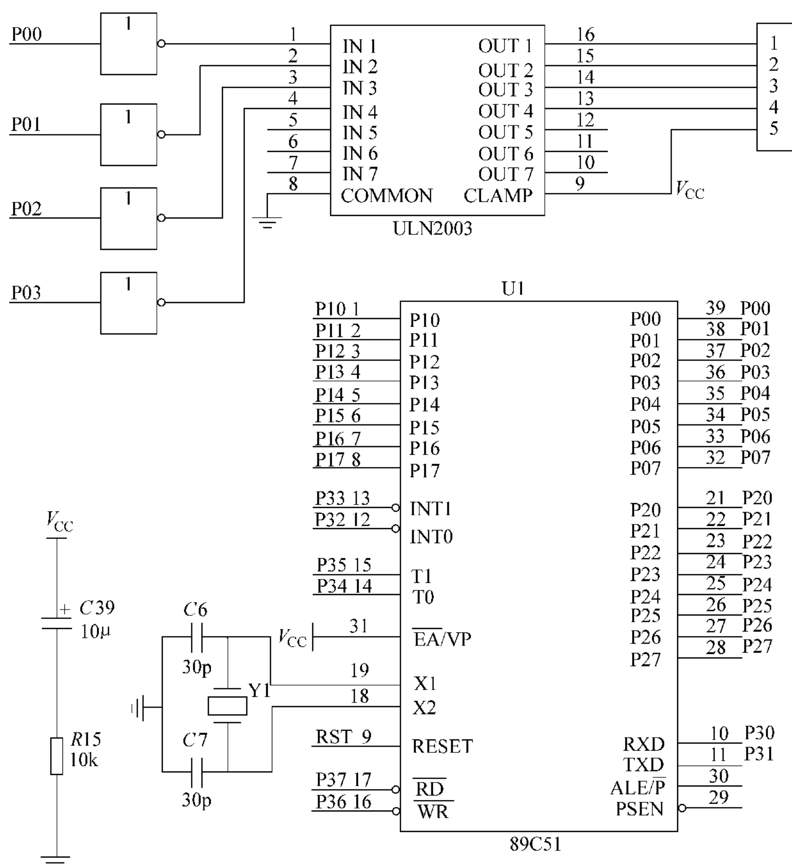


图 10-29　硬件原理

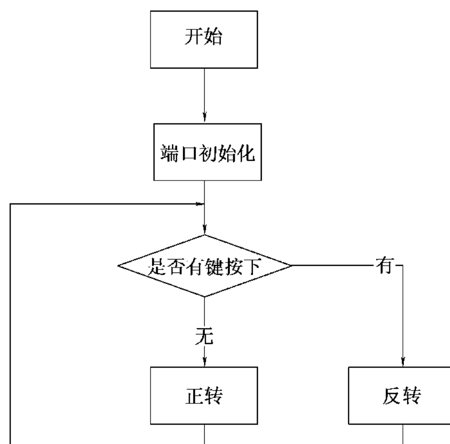


图 10-30　软件流程图

软件代码

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
/ *           杭州电子 & 计算机工作室                * /
/ *           http: //www. hificat. com              * /
```

```
/*          步进电机演示程序                  */
/*          目标器件：AT89C51                 */
/*          晶振：12MHz                       */
/*          编译环境：Keil 7.50A              */
/****************************/

/*********************包含头文件*********************/
#include <reg51.h>
/*********************端口定义*********************/
sbit key = P3^2;

/*******************************************
函数功能：延时子程序
入口参数：
出口参数：
*******************************************/
void delay (void)
{
    int k;
    for (k=0; k<2000; k++);
}

/*******************************************
函数功能：主程序
入口参数：
出口参数：
*******************************************/
void main ()
{
    P0=0x00;                        //输出全高
    key=1;                          //按键置输入状态
    while (1)                       //主循环
      {
        if (key==1)                 //无键按下正转
          {
           P0=0xFC;                 //1100
           delay ();
           P0=0xF6;                 //0110
           delay ();
```

```
            P0 = 0xF3;                  //0011
            delay ( );
            P0 = 0xF9;                  //1001
            delay ( );
        }
        else                            //有键按下反转
        {
            P0 = 0xFC;                  //1100
            delay ( );
            P0 = 0xF9;                  //1001
            delay ( );
            P0 = 0xF3;                  //0011
            delay ( );
            P0 = 0xF6;                  //0110
            delay ( );
        }
    }
}
```

10.4 I²C 总线器件应用实例

10.4.1 I²C 总线基本概念

I²C 总线，是 INTER INTEGRATED CIRCUIT BUS 的缩写，即“内部集成电路总线”。I²C 总线是 Philips 公司推出的一种双向二线制总线。目前 Philips 公司和其他集成电路制造商推出了很多基于 I²C 总线的外围器件。I²C 总线包括一条数据线（SDA）和一条时钟线（SCL）。协议允许总线接入多个器件，并支持多主工作。总线中的器件既可以作为主控器也可以作为被控器，既可以是发送器也可以是接收器。总线按照一定的通信协议进行数据交换。在每次数据交换开始，作为主控器的器件需要通过总线竞争获得主控权，并启动一次数据交换。系统中各个器件都具有唯一的地址，各器件之间通过寻址确定数据接收方。

10.4.2 I²C 总线的系统结构

一个典型的 I²C 总线标准的 IC 器件，其内部不仅有 I²C 接口电路，还可将内部各单元电路划分成若干相对独立的模块，它只有两根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL。CPU 可以通过指令对各功能模块进行控制。各种被控制电路均并联在这条总线上，但就像电话机一样只有拨通各自的号码才能工作，所以每个电路和模块都有唯一的地址，在信息的传输过程中，I²C 总线上并接的每一模块电路既是主控器（或被控器），又是发送器（或接收器）。CPU 发出的控制信号分为地址码和控制量（数据）两部分，地址码用来选址，即接通需要控制的电路，确定控制的种类；控制量决定该调整的类别及需要调整的

量。这样，各控制电路虽然挂在同一条总线上，却彼此独立，互不相关。I^2C 总线接口电路如图 10-31 所示。

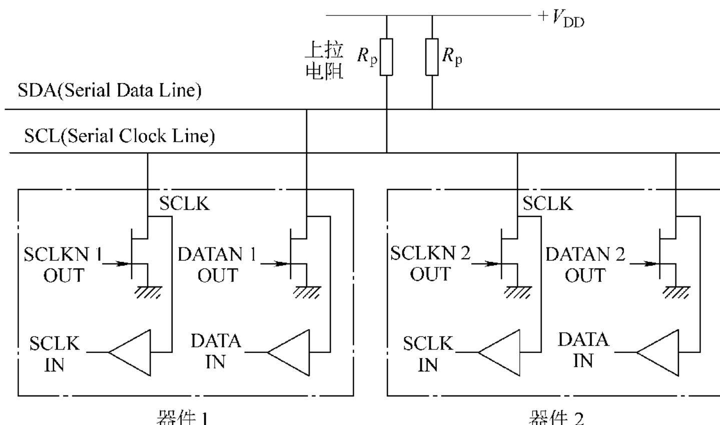


图 10-31　I^2C 总线的系统结构

I^2C 总线的器件分为主器件和从器件。主器件的功能是启动在总线上传送数据，并产生时钟脉冲，以允许与被寻址的器件进行数据传送。被寻址的器件，称为从器件。一般来讲，任何器件均可以成为从器件，只有微控制器才能称为主器件。主、从器件对偶出现，工作在接收还是发送数据方式，由器件的功能和数据传送方向所决定。

I^2C 总线允许连接多个微控制器，显然不能同时存在两个主器件，先控制总线的器件成为主器件，这就是总线竞争。在竞争过程中数据不会被破坏、丢失。数据只能在主、从器件中传送，结束后，主、从器件将释放总线，退出主、从器件角色。

10.4.3　I^2C 总线接口

传统的单片机串行接口的发送和接收一般都分别各用一条线，如 MCS-51 系列的 TXD 和 RXD，而 I^2C 总线则根据器件的功能通过软件程序使其工作于发送或接收方式。当某个器件向总线上发送信息时，它就是发送器（也叫主器件），而当其从总线上接收信息时，又成为接收器（也叫从器件）。主器件用于启动总线上传送数据并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。I^2C 总线的控制完全由挂在总线上的主器件送出的地址和数据决定，在总线上，既没有中心机也没有优先级。

总线上主和从（即发送和接收）的关系取决于此时数据传送的方向。SDA 和 SCL 都是双向线路，都通过一个电流源或上拉电阻连接到电源端。连接总线器件的输出级必须是集电极或漏极开路，以具有线“与”功能，当总线空闲时，两根线都是高电平。I^2C 总线上数据的传输速率在标准模式下可达 100Kbit/s，在快速模式下可达 400Kbit/s，在高速模式下可达 3.4Mbit/s 连接到总线的接口数量只由总线电容是 400pF 的限制决定。

10.4.4　I^2C 总线的时钟信号

在 I^2C 总线上传送信息时的时钟同步信号是由挂接在 SCL 时钟线上的所有器件的逻辑“与”完成的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号变为低电平，将使 SCL 线上所有器件开始并保持低电平期。此时，低电平周期短的器

件的时钟由低至高的跳变并不影响 SCL 线的状态，这些器件将进入高电平等待的状态。当所有器件的时钟信号都变为高电平时，低电平期结束，SCL 线被释放返回高电平，即所有的器件都同时开始它们的高电平期。其后，第一个结束高电平期的器件又将 SCL 线拉成低电平。这样就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件决定，而时钟高电平时间由时钟高电平期最短的器件决定。

10.4.5 I²C 总线的传输协议与数据传送

1. 起始和停止条件

在数据传送过程中，必须确认数据传送的开始和结束。在 I²C 总线技术规范中，开始和结束信号（也称启动和停止信号）的定义如图 10-32 所示。

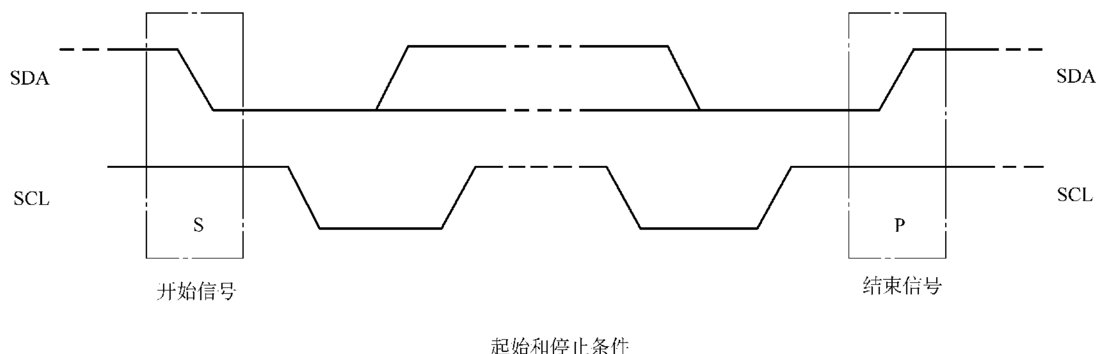


图 10-32　开始和结束信号时序

开始信号：当时钟总线 SCL 为高电平时，数据线 SDA 由高电平向低电平跳变，开始传送数据。

结束信号：当 SCL 线为高电平时，SDA 线从低电平向高电平跳变，结束传送数据。

开始和结束信号都是由主器件产生。在开始信号以后，总线即被认为处于忙状态，其他器件不能再产生开始信号。主器件在结束信号以后退出主器件角色，经过一段时间，总线被认为是空闲的。

2. 数据格式

I²C 总线数据传送采用时钟脉冲逐位串行传送方式，在 SCL 的低电平期间，SDA 线上高、低电平能变化，在高电平期间，SDA 上数据必须保持稳定，以便接收器采样接收，时序如图 10-33 所示。

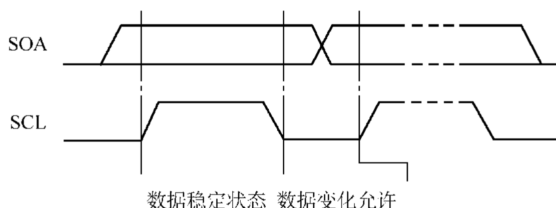


图 10-33　数据格式

I²C 总线发送器送到 SDA 线上的每个字节必须为 8 位长，传送时高位在前，低位在后。与之对应，主器件在 SCL 线上产生 8 个脉冲；第 9 个脉冲低电平期间，发送器释放 SDA 线，接收器把 SDA 线拉低，以给出一个接收确认位；第 9 个脉冲高电平期间，发送器收到这个确认位然后开始下一字节的传送，下一个字节的第 1 个脉冲低电平期间接收器释放 SDA。每个字节需要 9 个脉冲，每次传送的字节数是不受限制的。

I^2C 总线的数据传送格式是在 I^2C 总线开始信号后，送出的第一字节数据是用来选择从器件地址的，其中前 7 位为地址码，第 8 位为方向位（R/W）。方向位为“0”表示发送，即主器件把信息写到所选择的从器件中；方向位为“1”表示主器件将从从器件读信息。格式如下：

1	0	1	0	A2	A1	A0	R/W

注意：前 4 位固定为 1010。

开始信号后，系统中的各个器件将自己的地址和主器件送到总线上的地址进行比较，如果与主器件发送到总线上的地址一致，则该器件即被主器件寻址的器件，其接收信息还是发送信息则由第 8 位（R/W）决定。发送完第一个字节后再开始发数据信号。

3. 响应

数据传输必须带响应。相关的响应时钟脉冲由主机产生，当主器件发送完一字节的数据后，接着发出对应于 SCL 线上的一个时钟（ACK）认可位，此时钟内主器件释放 SDA 线，一字节传送结束，而从器件的响应信号将 SDA 线拉成低电平，使 SDA 在该时钟的高电平期间为稳定的低电平。从器件的响应信号结束后，SDA 线返回高电平，进入下一个传送周期。通常被寻址的接收器在接收到的每个字节后必须产生一个响应。当从机不能响应从机地址时，从机必须使数据线保持高电平，主机然后产生一个停止条件终止传输或者产生重复起始条件开始新的传输。如果从机接收器响应了从机地址但是在传输了一段时间后不能接收更多数据字节，主机必须再一次终止传输。这个情况用从机在第一个字节后没有产生响应来表示。从机使数据线保持高电平主机产生一个停止或重复起始条件。完整的数据传送过程如图 10-34 所示。

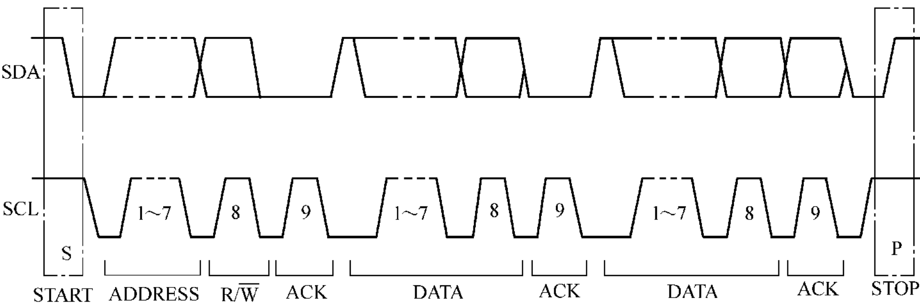


图 10-34　数据传送时序

I^2C 总线还具有广播呼叫地址用于寻址总线上所有器件的功能。若一个器件不需要广播呼叫寻址中所提供的任何数据，则可以忽略该地址不作响应。如果该器件需要广播呼叫寻址中按需提供的数据，则应对地址作出响应，其表现为一个接收器。

10.4.6　I^2C 总线接口器件应用

下面以目前在单片机系统中常用的带 I^2C 接口的 EEPROM 芯片 AT24C01 为例，介绍 I^2C 器件的基本应用。

1. AT24C01 简介

AT24C01 是美国 ATMEL 公司的低功耗 CMOS 串行 EEPROM，它内含 256 ×8 位存储空

间，具有工作电压宽（2.5～5.5V）、擦写次数多（>10000次）、写入速度快（<10ms）等特点。AT24C01中带有片内寻址寄存器。每写入或读出一个数据字节后，该地址寄存器自动加1，以实现对下一个存储单元的操作。所有字节都以单一操作方式读取。为降低总的写入时间，一次操作可写入多达8字节的数据。图10-35为AT24C01的封装图。各引脚功能如下：

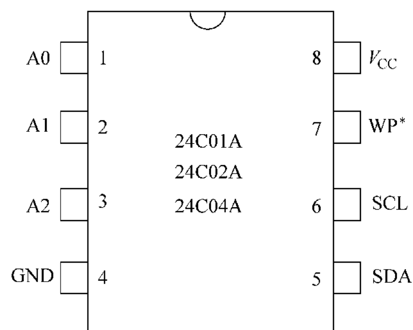


图10-35　AT24C01封装图

SCL：串行时钟。在该引脚的上升沿时，系统将数据输入到每个EEPROM器件，在下降沿时输出。

SDA：串行数据。该引脚为开漏极驱动，可双向传送数据。

A0、A1、A2：器件/页面寻址。为器件地址输入端。

WP：硬件写保护。当该引脚为高电平时禁止写入，为低电平时可正常读写数据。

V_{CC}：电源。一般输入+5V电压。

GND：接地。

2. AT24C01与单片机硬件图

图10-36为AT24C01与单片机的一种应用实例，图中AT24C01的引脚1、2、3是三条地址线，用于确定芯片的硬件地址。在AT89C51试验板上它们都接地，引脚8和引脚4分别为正、负电源。引脚5SDA为串行数据输入/输出，数据通过这条双向I^2C总线串行传送，在AT89C51试验板上和单片机的P11连接。引脚6SCL为串行时钟输入线，在AT89C51试验板上和单片机的P10连接。SDA和SCL都需要和正电源间各接一个上拉电阻。引脚7为写保护端，此处需要接地。

3. 程序设计

1）程序功能：本程序用来验证对AT24C01的读写是否正确，具体实现方法如下：上电初始化时向所有空间填充0xFF，再向01和02地址写两个数，通过程序对某个地址读数并显示，从而达到验证读写是否正确。

2）程序代码：本例的程序功能是实现利用单片机P10、P11作为SCL、SDA串行通信线，实现对某一地址内数据的读写校验操作。

```
/******************************/
/*        杭州电子&计算机工作室          */
/*        http://www.hificat.com         */
/*        EEPROM读写演示程序             */
/*        目标器件：AT89C51              */
/*        晶振：12MHz                    */
/*        编译环境：Keil 7.50A           */
/******************************/
#include <reg51.h>
#include <intrins.h>
```

图 10-36　51 单片机与 AT24C01 的连接

```
#define OP_READ 0xa1          // 器件地址以及读取操作
#define OP_WRITE 0xa0         // 器件地址以及写入操作

unsigned char code tab [ ] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80,
0x90};

int k;

sbit SDA = P1^1;
sbit SCL = P1^0;

/*******************************************
函数功能：延时子程序
入口参数：ms
出口参数：
*******************************************/
```

```
void delayms (unsigned char ms)
{
    unsigned char i;
    while (ms--)
     {
      for (i = 0; i < 120; i++);
     }
}

/****************************************************
函数功能：延时子程序
入口参数：
出口参数：
****************************************************/
void delay (void)
{
    int k;
    for (k=0; k<600; k++);
}

/****************************************************
函数功能：开始信号
入口参数：
出口参数：
****************************************************/
void start ()
{
    SDA = 1;
    SCL = 1;
    _nop_ ();
    _nop_ ();
    SDA = 0;
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    SCL = 0;
}
```

```
/*****************************************
函数功能：停止信号
入口参数：
出口参数：
*****************************************/
void stop ( )
{
    SDA = 0;
    _nop_ ( );
    _nop_ ( );
    SCL = 1;
    _nop_ ( );
    _nop_ ( );
    _nop_ ( );
    _nop_ ( );
    SDA = 1;
}

/*****************************************
函数功能：读取数据
入口参数：
出口参数：read_data
*****************************************/
unsigned char shin ( )

{
    unsigned char i, read_data;
    for (i = 0; i < 8; i++)
    {
        SCL = 1;
        read_data <<= 1;
        read_data |= (unsigned char) SDA;
        SCL = 0;
    }
    return (read_data);
}

/*****************************************
```

```
函数功能：向 EEPROM 写数据
入口参数：write_data
出口参数：ack_bit
*******************************************/
bit shout (unsigned char write_data)
{
    unsigned char i;
    bit ack_bit;
    for (i = 0; i < 8; i++)              // 循环移入 8 个位
    {
      SDA = (bit) (write_data & 0x80);
      _nop_ ();
      SCL = 1;
      _nop_ ();
      _nop_ ();
      SCL = 0;
      write_data <<= 1;
    }
    SDA = 1;                             // 读取应答
    _nop_ ();
    _nop_ ();
    SCL = 1;
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    ack_bit = SDA;
    SCL = 0;
    return ack_bit;                      // 返回 AT24Cxx 应答位
}

/*******************************************
函数功能：向指定地址写数据
入口参数：addr，write_data
出口参数：
*******************************************/
void write_byte (unsigned char addr, unsigned char write_data)
{
```

```
    start ();
    shout (OP_WRITE);
    shout (addr);
    shout (write_data);
    stop ();
    delayms (10);
}

/****************************************************
函数功能：读取当前地址数据
入口参数：
出口参数：read_data
****************************************************/
unsigned char read_current ()
{
    unsigned char read_data;
    start ();
    shout (OP_READ);
    read_data = shin ();
    stop ();
    return read_data;
}

/****************************************************
函数功能：向指定地址读数据
入口参数：random_addr
出口参数：read_data
****************************************************/
unsigned char read_random (unsigned char random_addr)
{
    start ();
    shout (OP_WRITE);
    shout (random_addr);
    return (read_current ());
}

/****************************************************
函数功能：显示子程序
```

```
入口参数：k
出口参数：
*****************************************/
void display (int k)
{
    P2 =0xfe;
    P0 =tab [k/1000];
    delay ();
    P2 =0xfd;
    P0 =tab [k%1000/100];
    delay ();
    P2 =0xfb;
    P0 =tab [k%100/10];
    delay ();
    P2 =0xf7;
    P0 =tab [k%10];
    delay ();
    P2 =0xff;
}

/*****************************************
函数功能：主程序
入口参数：
出口参数：
*****************************************/
main (void)
{
    SDA  =  1;
    SCL  =  1;
    k =0;
    write_byte (0x01, 0x22);
    delayms (250);
    write_byte (0x02, 0x33);
    delayms (250);
    delayms (250);
    k =  read_random (0x01);
    delayms (250);
    while (1)
    {
```

```
    display (k);
    }
}
```

10.5 93CXX 系列存储器应用实例

10.4 节介绍了 I^2C 总线结构的存储器 24CXX 的应用，在一般的存储器件中 SPI 结构的存储器也用得比较广泛，本节介绍 SPI 总线结构的存储器 93CXX 的应用。

10.5.1 SPI 总线简介

1. SPI 总线基本概念

SPI（ Serial Peripheral Interface，串行外设接口）总线是 Motorola 公司推出的一种同步串行接口技术。SPI 总线系统是一种同步串行外设接口，允许 MCU 与各种外围设备以串行方式进行通信、数据交换。外围设备包括 FLASHRAM、A/ D 转换器、网络控制器、MCU 等。SPI 是一种高速的、全双工、同步的通信总线，并且在芯片的引脚上只占用 4 根线，节约了芯片的引脚，同时为 PCB 的布局节省了空间，提供了方便，正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议。其工作模式有两种：主模式和从模式。SPI 是一种允许一个主设备启动一个从设备的同步通信的协议，从而完成数据的交换。也就是 SPI 是一种规定好的通信方式。这种通信方式的优点是占用端口较少，一般 4 根线就够基本通信了（不算电源线）。同时传输速度也很高。一般来说要求主设备要有 SPI 控制器（也可用模拟方式），就可以与基于 SPI 的芯片通信了。

2. SPI 总线系统结构

SPI 系统可直接与各个厂家生产的多种标准外围器件直接接口，一般使用 4 根线：串行时钟线（SCK）、主机输入/ 从机输出数据线 MISO（DO）、主机输出/ 从机输入数据线 MOSI（DI）和低电平有效的从机选择线 CS。MISO 和 MOSI 用于串行接收和发送数据，先为 MSB（高位），后为 LSB（低位）。在 SPI 设置为主机方式时，MISO 是主机数据输入线，MOSI 是主机数据输出线。SCK 用于提供时钟脉冲将数据一位位地传送。SPI 总线器件间传送数据框图如图 10-37 所示。

3. SPI 总线的接口特性

利用 SPI 总线可在软件的控制下构成各种系统。如 1 个主 MCU 和几个从 MCU、几个从 MCU 相互连接构成多主机系统（分布式系统）、1 个主 MCU 和 1 个或几个从 I/O 设备所构成的各种系统等。在大多数应用场合，可使用 1 个 MCU 作为主控机来控制数据，并向 1 个或几个从外围器件传送该数据。从器件只有在主机发命令时才能接收或发送数据。其数据的传输格式是高位（MSB）在前，低位（LSB）在后。

当一个主控机通过 SPI 与几种不同的串行 I/O 芯片相连时，必须使用每片的允许控制端，这可以通过 MCU 的 I/O 端口输出线来实现。但应特别注意这些串行 I/O 芯片的输入输出特性：首先是输入芯片的串行数据输出是否有三态控制端。平时未选中芯片时，输出端应处于高阻态。若没有三态控制端，则应外加三态门。否则 MCU 的 MISO 端只能连接 1 个输入芯片。其次是输出芯片的串行数据输入是否有允许控制端。因为只有在此芯片允许时，

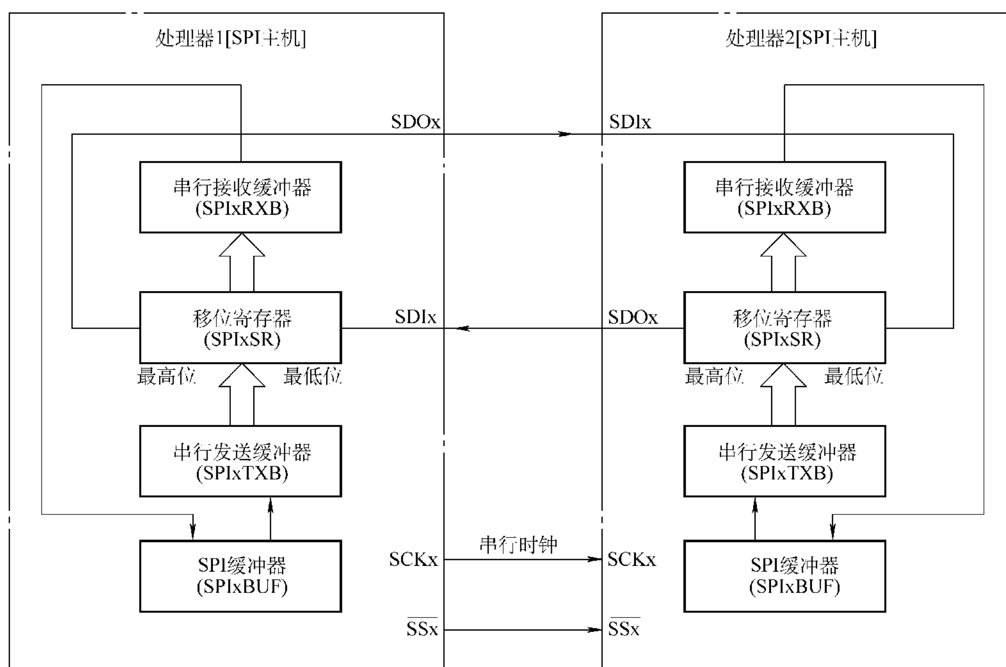


图 10-37　SPI 总线器件间传送数据框图

SCK 脉冲才把串行数据移入该芯片；在禁止时，SCK 对芯片无影响。若没有允许控制端，则应在外围用门电路对 SCK 进行控制，然后再加到芯片的时钟输入端；当然，也可以只在 SPI 总线上连接 1 个芯片，而不再连接其他输入或输出芯片。

4. SPI 总线的数据传输

SPI 是一个环形总线结构，其时序其实很简单，主要是在 SCK 的控制下，两个双向移位寄存器进行数据交换。SPI 数据传输原理很简单，它需要至少 4 根线，事实上 3 根线也可以。也是所有基于 SPI 的设备共有的，它们是 SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选）。其中 CS 是控制芯片是否被选中的，也就是说只有片选信号为预先规定的使能信号时（高电位或低电位），对此芯片的操作才有效。这就允许在同一总线上连接多个 SPI 设备成为可能。在 SPI 方式下数据是一位一位地传输的。这就是 SCK 时钟线存在的原因，由 SCK 提供时钟脉冲，SDI，SDO 则基于此脉冲完成数据传输。数据输出通过 SDO 线，数据在时钟上沿或下沿时改变，在紧接着的下沿或上沿被读取。完成一位数据传输，输入也使用同样原理。这样，在至少 8 次时钟信号的改变（上沿和下沿为一次），就可以完成 8 位数据的传输。假设 8 位寄存器内装的是待发送的数据 10101010，上升沿发送、下降沿接收、高位先发送。那么第一个上升沿来的时候数据将会是高位数据 SDO = 1。下降沿到来的时候，SDI 上的电平将被存到寄存器中去，那么这时寄存器 = 0101010SDI，这样在 8 个时钟脉冲以后，两个寄存器的内容互相交换一次。这样就完成里一个 SPI 时序。下面举一个实例来说明其数据传送过程。

假设主机和从机初始化就绪，并且主机的 sbuff = 0xaa，从机的 sbuff = 0x55，下面将分步对 SPI 的 8 个时钟周期的数据情况演示一遍：（下表中“上”表示上升沿，“下”表示下降沿）见表 10-9。

表 10-9 脉冲与数据变化对应表

脉冲序号	主机缓存	从机缓存	SDI	SDO
0	10101010	01010101	0	0
1 上	0101010x	1010101x	0	1
1 下	01010100	10101011	0	1
2 上	1010100x	0101011x	1	0
2 下	10101001	01010110	1	0
3 上	0101001x	1010110x	0	1
3 下	01010010	10101101	0	1
4 上	1010010x	0101101x	1	0
4 下	10100101	01011010	1	0
5 上	0100101x	1011010x	0	1
5 下	01001010	10110101	0	1
6 上	1001010x	0110101x	1	0
6 下	10010101	01101010	1	0
7 上	0010101x	1101010x	0	1
7 下	00101010	11010101	0	1
8 上	0101010x	1010101x	1	0
8 下	01010101	10101010	1	0

这样就完成了两个寄存器 8 位的交换，SDI、SDO 是相对于主机而言的。其中 CS 引脚作为主机的时候，从机可以把它拉低被动选为从机，作为从机的是时候，可以作为片选脚用。根据以上分析，一个完整的传送周期是 16 位，即两个字节，因为首先主机要发送命令，然后从机根据主机的命令准备数据，主机在下一个 8 位时钟周期才把数据读回来。这样的传输方式有一个优点，与普通的串行通信不同，普通的串行通信一次连续传送至少 8 位数据，而 SPI 允许数据一位一位地传送，甚至允许暂停，因为 SCK 时钟线由主控设备控制，当没有时钟跳变时，从设备不采集或传送数据。也就是说，主设备通过对 SCK 时钟线的控制可以完成对通信的控制。SPI 还是一个数据交换协议，因为 SPI 的数据输入和输出线独立，所以允许同时完成数据的输入和输出。

对于不带 SPI 串行总线接口的 MCS51 系列单片机来说，可以使用软件来模拟 SPI 的操作，包括串行时钟、数据输入和数据输出。如我们可以定义三个普通 I/O 口用来模拟 SPI 器件的 SCK、MISO、MOSI。对于不同的串行接口外围芯片，它们的时钟时序是不同的。对于在 SCK 的上升沿输入（接收）数据和在下降沿输出（发送）数据的器件，一般应将其串行时钟输出口的初始状态设置为 1，而在允许接口后再置为 0。这样，MCU 在输出 1 位 SCK 时钟的同时，将使接口芯片串行左移，从而输出 1 位数据至单片机的模拟 MISO 线，此后再置 SCK 为 1，使单片机从模拟的 MOSI 线输出 1 位数据（先为高位）至串行接口芯片。至此，模拟 1 位数据输入输出便宣告完成。此后再置 SCK 为 0，模拟下 1 位数据的输入输出……，依此循环 8 次，即可完成 1 次通过 SPI 总线传输 8 位数据的操作。对于在 SCK 的下降沿输入

数据和上升沿输出数据的器件，则应取串行时钟输出的初始状态为 0，即在接口芯片允许时，先置 SCK 为 1，以便外围接口芯片输出 1 位数据（MCU 接收 1 位数据），之后再置时钟为 0，使外围接口芯片接收 1 位数据（MCU 发送 1 位数据），从而完成 1 位数据的传送。

10.5.2 93C46 存储器的软硬件设计实例

下面就以目前单片机系统中应用广泛的 SPI 接口的数据存储器 93C46 为例，介绍 SPI 器件的基本应用。

1. 93C46 串行存储器简介

93C46 是 1k 位串行 EEPROM 储存器。每一个储存器都可以通过 DI/DO 引脚写入或读出。它的存储容量为 1024 位，内部为 128×8 位或 64×16 位。93C46 为串行三线 SPI 操作芯片，在时钟时序的同步下接收数据口的指令。指令码为 9 位十进制码，具有 7 个指令，读、擦写使能、擦除、写、全擦、全写及擦除禁止。该芯片擦写时间快，有擦写使能保护，可靠性高，擦写次数可达 100 万次，93C46 的引脚功能如图 10-38 所示。93C46 串行 EEPROM 指令格式选择表见表 10-10。

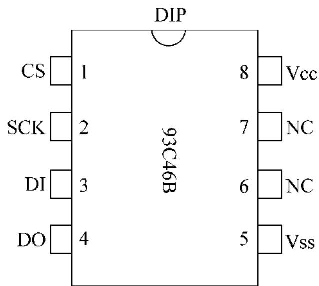


图 10-38 93C46 的引脚图

CS-芯片选择 SCK-时钟 DI-串行数据输入 DO-串行数据输出 VSS-接地 NC-空脚（应用时不用接任何电路） VCC-电源

表 10-10 93C46 串行 EEPROM 指令格式选择表

指令	起始位	操作数	地址		数据	
			64×16	128×8	64×16	128×8
读（READ）	1	10	A5～A0	A6～A0		
清除（ERASE）	1	11	A5～A0	A6～A0		
写（WRITE）	1	01	A5～A0	A6～A0	D15～D0	D7～D0
写使能（EWEN）	1	00	11XXXX	11XXXXX		
写禁止（EWDS）	1	00	00XXXX	00XXXXX		
芯片清除（ERAL）	1	00	10XXXX	10XXXXX		
芯片写入（WRAL）	1	00	01XXXX	01XXXXX	D15～D0	D7～D0

指令说明：

1）读（READ）：当下达 10XXXXXX 指令后，地址（XXXXXXXX）的数据在 SCK＝1 时由 DO 输出。

2）写（WRITE）：在写入数据前，必须先下达写使能（EWEN）指令，然后再下达 01XXXXXX 指令后，当 SCK＝1 时，会把数据码写入指定地址（XXXXXXXX）；而 DO＝0 时，表示还在进行写操作，写入结束后 DO 会转为高电平。写入动作完成后，必须再下达写禁止（EWDS）命令。

3）清除（ERASE）：下达清除指令 11XXXXXX 后会将地址（XXXXXXXX）的数据清除。

4）写使能（EWEN）：下达 0011XXXX 指令后，才可以进行写（WRITE）操作。

5）写禁止（EWDS）：下达 0000XXXX 指令后，才可重复进行写入（WRITE）操作。

6）芯片清除（ERAL）：下达 0010XXXX 指令后，全部禁止。

7）芯片写入（WRAL）：下达 0001XXXX 指令后，全部写入“0”。

2. 程序功能

本例用来实现对 93C46 存储器的读写操作，并验证数据是否正确。此程序可以用配套实验板与 KC-106 单片机总线模块相连接作为硬件实验平台。本程序先分别向 0x02 和 0x03 两个地址写入 0x55 和 0xAA，然后读其中一个地址，并将读到的数据显示出来验证是否正确。程序默认是读 0x02 地址内的数据，读者也可以修改地址数据来读其他地址数据。实验时，请使用 KC-106 单片机总线模块进行扩展，从而完成 SPI 总线的实验。

3. 硬件原理图

如图 10-39 所示。

图 10-39　硬件原理图

4. 程序流程图

见图10-40所示。

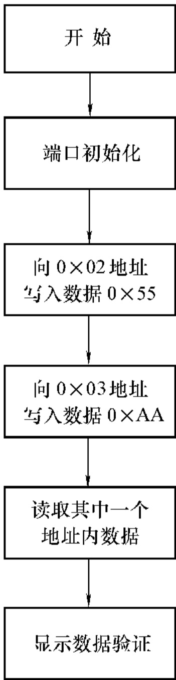


图10-40 软件流程图

5. 软件代码

```
/*****************************************/
/*杭州晶控电子有限公司                        */
/*http://www.hificat.com                    */
/*93C46 演示程序                              */
/*目标器件:AT89S51                           */
/*晶振:11.0592MHz                            */
/*编译环境:Keil 7.50A                        */
/****************************************

/********************包含头文件*******************/
#include <reg51.h>
#include <intrins.h>

/*********************数据定义********************/
#define OP_EWEN_H      0x00     // 00          write enable
#define OP_EWEN_L      0x60     // 11X XXXX    write enable
#define OP_EWDS_H      0x00     // 00          disable
#define OP_EWDS_L      0x00     // 00X XXXX    disable
```

```
#define OP_WRITE_H          0x40        // 01 A6-A0         write data
#define OP_READ_H           0x80        // 10 A6-A0         read data
#define OP_ERASE_H          0xc0        // 11 A6-A0         erase a word
#define OP_ERAL_H           0x00        // 00               erase all
#define OP_ERAL_L           0x40        // 10X XXXX         erase all
#define OP_WRAL_H           0x00        // 00               write all
#define OP_WRAL_L           0x20        // 01X XXXX         write all

/*******************端口定义*****************/
sbit CS  = P3^4;
sbit SK  = P3^3;
sbit DI     = P3^5;
sbit DO  = P3^6;
/****************共阳LED段码表****************/
unsigned char code tab[ ] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

/****************定义全局变量*********************/
int readdata;                          //从93C46读出的数据

/*****************************************************
函数功能:数码管扫描延时子程序
入口参数:
出口参数:
*****************************************************
void delay1(void)
{
    int k;
    for(k=0;k<400;k++);
}

/*****************************************************
函数功能:读写延时子程序
入口参数:ms
出口参数:
*****************************************************/
void delayms(unsigned char ms)
{
    unsigned char i;
    while(ms--)
```

```
    {
        for(i = 0; i < 120; i++);
    }
}

/*******************************************************
函数功能:数码管显示子程序
入口参数:k
出口参数:
*******************************************************/
void display(int k)
{
    P2 = 0xfe;
    P0 = tab[k/1000];
    delay1();
    P2 = 0xfd;
    P0 = tab[k%1000/100];
    delay1();
    P2 = 0xfb;
    P0 = tab[k%100/10];
    delay1();
    P2 = 0xf7;
    P0 = tab[k%10];
    delay1();
    P2 = 0xff;
}

/*******************************************************
函数功能:写入指令和地址子程序
入口参数:op_h,op_l
出口参数:
*******************************************************/
void inop(unsigned char op_h, unsigned char op_l)
{
    unsigned char i;
    SK = 0;                    // 开始位
    DI = 1;
    CS = 1;
    _nop_();
```

```
    _nop_();
    SK = 1;
    _nop_();
    _nop_();
    SK = 0;                          // 开始位结束
    DI = (bit)(op_h & 0x80);         // 移入指令码高位
    SK = 1;
    op_h <<= 1;
    SK = 0;
    DI = (bit)(op_h & 0x80);         // 移入指令码低位
    SK = 1;
    _nop_();
    _nop_();
    SK = 0;
    op_l <<= 1;                      // 移入余下的指令码或地址数据
    for(i = 0; i < 7; i++)
    {
        DI = (bit)(op_l & 0x80);     // 先移入高位
        SK = 1;
        op_l <<= 1;
        SK = 0;
    }
    DI = 1;
}

/*****************************************
函数功能:写入数据子程序
入口参数:indata
出口参数:
*****************************************/
void shin(unsigned char indata)
{
    unsigned char i;
    for(i = 0; i < 8; i++)
    {
        DI = (bit)(indata & 0x80);     // 先移入高位
        SK = 1;
        indata <<= 1;
        SK = 0;
```

```
    }
    DI = 1;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能:写入数据使能子程序
入口参数:
出口参数:
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void ewen( )
{
    inop( OP_EWEN_H, OP_EWEN_L);
    CS = 0;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能:写入数据禁止子程序
入口参数:
出口参数:
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void ewds( )
{
    inop( OP_EWDS_H, OP_EWDS_L);
    CS = 0;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能:数据清除子程序
入口参数:
出口参数:
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void erase( )
{
    inop( OP_ERAL_H, OP_ERAL_L);
    delayms(30);
    CS = 0;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
函数功能:写入数据子程序
入口参数:addr,indata
出口参数:
*******************************************/
void write(unsigned char addr, unsigned char indata)
{
    inop(OP_WRITE_H, addr);      //写入指令和地址
    shin(indata);                //写入数据
    CS = 0;
    delayms(10);
}

/*******************************************
函数功能:读出数据子程序
入口参数:
出口参数:outdata
*******************************************/
unsigned char shout(void)
{
    unsigned char i, out_data;
    for(i = 0; i < 8; i++)
    {
        SK = 1;
        out_data <<= 1;
        SK = 0;
        out_data |= (unsigned char)DO;
    }
    return(out_data);
}

/*******************************************
函数功能:读出某地址数据子程序
入口参数:addr
出口参数:out_data
*******************************************/
unsigned char read(unsigned char addr)
{
    unsigned char out_data;
    inop(OP_READ_H, addr);
```

```
    out_data = shout();
    CS = 0;
    return out_data;
}

/****************************************************
函数功能:主程序
入口参数:
出口参数:
****************************************************/
void main(void)
{
    unsigned char i;
    CS = 0;                          //初始化端口
    SK = 0;
    DI = 1;
    DO = 1;
    ewen();                          //使能写入操作
    erase();                         //擦除全部内容
    write(0x02, 0x55);               //向 0x02 地址写入 0x55(85)
    write(0x03, 0xAA);               //向 0x03 地址写入 0xAA(170)
    while(1)
    {
        readdata = read(0x02);       //读取其中一个地址内数据验证
        display(readdata);           //显示数据
    }
}
```

10.6 DS18B20 数字温度传感器应用实例

随着现代化信息技术的飞速发展和传统工业改造的逐步实现，能独立工作的温度检测系统已广泛应用于各种不同领域。传统的温度检测系统大多采用热敏电阻作为传感器。采用热敏电阻作为传感器的温度检测系统必须经过专门的接口电路转换成数字信号后才能由微处理器进行处理，存在可靠性差、成本高、精度低等诸多缺点。现在很多温度检控场合已广泛使用单总线的温度传感器，使整个系统简单可靠。

10.6.1 单总线（1-WIRE）技术介绍

目前单片机外设的接口形式主要有单总线接口、I^2C 接口、SPI 接口、PS2 接口等。SPI 接口与单片机通信需要三根线，I^2C 接口也要两根线，而单总线器件与单片机间数据通信只

要一根线。美国 DALLAS 公司推出的单总线（1-WIRE BUS）技术与 I^2C、SPI、PS2 总线不同，它采用单根信号线，既可以传输时钟信号又可以传送数据信号，而数据又可双向传送，因而这种总线技术具有线路简单、成本低廉、便于扩展和维护等优点。

单总线适用于单主机系统，能够控制一个或多个从机设备。主机可以是微处理器，从机可以是单总线器件，它们之间的数据交换只通过一条信号线。当只有一个从机设备时，系统可按单节点系统操作；当有多个从设备时，系统则按多节点系统操作。主机或从机通过一个漏极开路或三态端口连接到这个数据线，以允许设备在不发送数据时能够释放总线，而让其他设备使用总线，其内部等效电路如图 10-41 所示。单总线通常要求接一个约为 4.7kΩ 左右的上拉电阻，这样，当总线空闲时，其状态为高电平。主机和从机之间的通信可以通过三个步骤完成，分别是初始化单总线器件、识别单总线器件、数据交换。由于它们是主从结构，只有主机呼号从机时，从机才能应答，因此主机访问单总线器件时都必须严格遵循单总线命令序列。如果出现序列混乱，单总线器件将不响应主机。

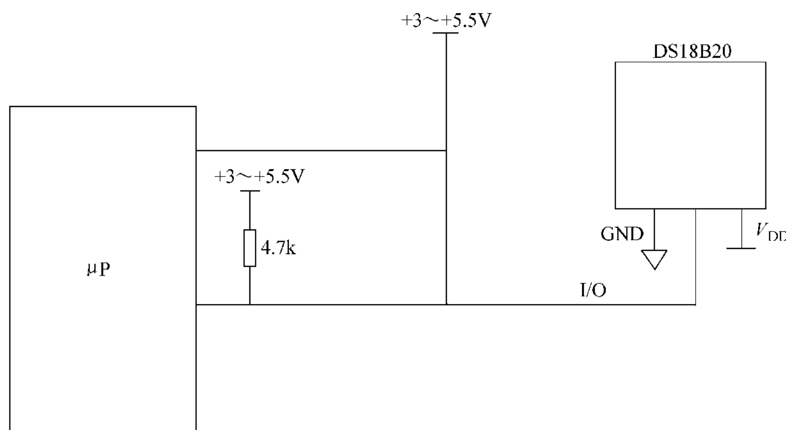


图 10-41　等效电路

所有的单总线器件都要遵循严格的通信协议，以保证数据的完整性。单总线协议定义了复位信号、应答信号、写“0”、读“0”、写“1”、读“1”的几种时序信号类型。所有的单总线命令序列都是由这些基本的信号类型组成的。在这些信号中，除了应答脉冲外，其他均由主机发出同步信号，并且发送的所有命令和数据都是字节的低位在前。

所有单总线器件的读、写时序至少需要 60μs，且每两个独立的时序间至少需要 1μs 的恢复时间。在写时序中，主机将在拉低总线 15μs 之内释放总线，并向单总线器件写“1”；如果主机拉低总线后能保持至少 60μs 的低电平，则向总线器件写“0”。单总线器件仅在主机发出读时序时才向主机传输数据，所以，当主机向单总线器件发出读数据命令后，必须马上产生读时序，以便单总线器件能传输数据。

10.6.2　DS18B20 简介

在多点温度测量系统中，单总线数字温度传感器因其体积小、构成的系统结构简单等优点，应用越来越广泛。每一个数字温度传感器内均有唯一的 64 位序列号（最低 8 位是产品代码，其后 48 位是器件序列号，最后 8 位是前 56 位循环冗余校验码），只有获得该序列号后才可能对其进行操作，也才能在多传感器系统中将它们一一识别。

DS18B20 是 DALLAS 公司生产的一线式数字温度传感器，它具有微型化、低功耗、高性能、抗干扰能力强、易配处理器等优点，特别适用于构成多点温度测控系统，可直接将温度转化成串行数字信号（提供 9 位二进制数字）给单片机处理，且在同一总线上可以挂接多个传感器芯片。它具有 3 引脚 TO-92 小体积封装形式，温度测量范围为 -55 ~ +125℃，可编程为 9 ~ 12 位 A/D 转换精度，测温分辨率可达 0.0625℃，被测温度用符号扩展的 16 位数字量方式串行输出，其工作电源既可在远端引入，也可采用寄生电源方式产生，多个 DS18B20 可以并联到三根或两根线上，CPU 只需一根端口线就能与多个 DS18B20 通信，占用微处理器的端口较少，可节省大量的引线和逻辑电路。以上特点使 DS18B20 非常适用于远距离多点温度检测系统。

10.6.3 DS18B20 新性能

独特的单线接口仅需一个端口引脚进行通信。

简单的多点分布应用。

无需外部器件。

可通过数据线供电。

零待机功耗。

测温范围 -55 ~ +125℃，以 0.5℃递增。华氏器件 -67 ~ +257°F，以 0.9°F 递增。

可编程的分辨率为 9 ~ 12 位，对应的可分辨温度分别为 0.5℃、0.25℃、0.125℃ 和 0.0625℃。

温度数字量转换时间 200ms（典型值），12 位分辨率时最多在 750ms 内把温度值转换为数字。

用户可定义的非易失性温度报警设置。

报警搜索命令识别并标志超过程序限定温度（温度报警条件）的器件。

应用包括温度控制、工业系统、消费品、温度计或任何热感测系统。

负压特性：电源极性接反时，温度计不会因发热而烧毁，但不能正常工作。

10.6.4 DS18B20 外形及引脚说明

外形及引脚说明如图 10-42 所示。

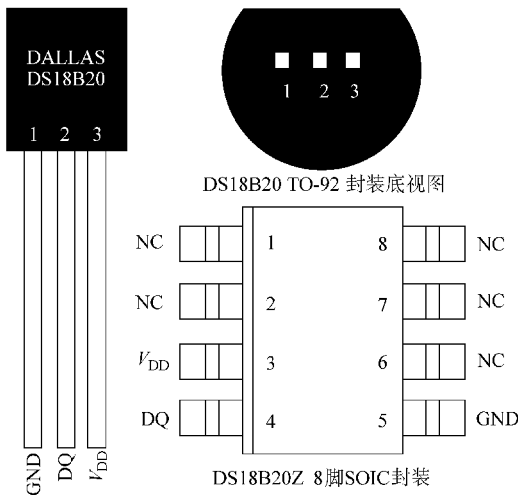


图 10-42 DS18B20 的引脚排列
1—GND：地 2—DQ：单线运用的数据输入/输出引脚 3—V_{DD}：可选的电源引脚

10.6.5 DS18B20 内特性

DS18B20 内部结构如图 10-43 所示，主要由 4 部分组成：64 位 ROM、温度敏感元件、非易失性温度报警触发器 TH 和 TL、配置寄存器。ROM 中的 64 位序列号是出厂前被光刻好的，它可以看作是该 DS18B20 的地址序列码，每个 DS18B20 的 64 位序列号均不相同。64 位激光 ROM 从高位到低位依次

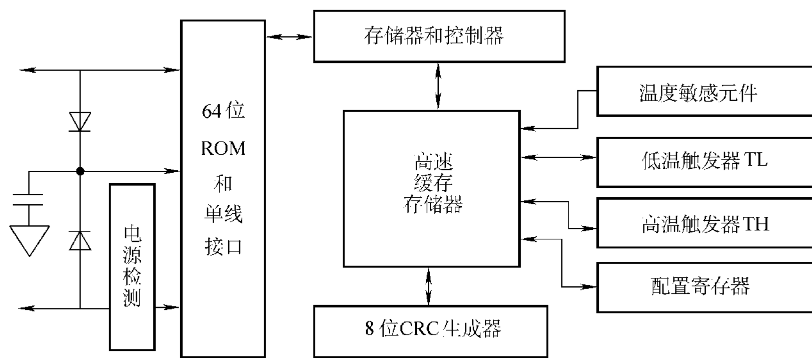


图 10-43　DS18B20 内部结构框图

为 8 位 CRC、48 位序列号和 8 位家族代码（28H）组成。ROM 的作用是使每一个 DS18B20 都各不相同，这样就可以实现一根总线上挂接多个 DS18B20 的目的。非易失性温度报警触发器 TH 和 TL 可通过软件写入用户报警上下限值。配置寄存器为高速暂存存储器中的第 5 个字节。DS18B20 在工作时按此寄存器中的分辨率将温度转换成相应精度的数值，其各位定义如图 10-44 所示。其中，TM 为测试模式标志位，出厂时被写入 0，不能改变；R0、R1 为温度计分辨率设置位，其对应四种分辨率见配置寄存器与分辨率关系表。出厂时 R0、R1 置为缺省值：R0 = 1，R1 = 1（即 12 位分辨率），用户可根据需要改写配置寄存器以获得合适的分辨率，配置寄存器与分辨率的关系见表 10-11。



TM	R1	R0	1	1	1	1	1

MSB　　　　LSB

图 10-44　寄存器定义

表 10-11　配置寄存器与分辨率关系表

R0	R1	温度计分辨率/bit	最大转换时间/ms
0	0	9	93.75
0	1	10	187.5
1	0	11	375
1	1	12	750

DS18B20 工作过程及时序：

DS18B20 内部的低温度系数振荡器是一个振荡频率随温度变化很小的振荡器，为计数器 1 提供一频率稳定的计数脉冲。

高温度系数振荡器是一个振荡频率对温度很敏感的振荡器，为计数器 2 提供一个频率随温度变化的计数脉冲。

初始时，温度寄存器被预置成 −55℃，每当计数器 1 从预置数开始减计数到 0 时，温度寄存器中寄存的温度值就增加 1℃，这个过程重复进行，直到计数器 2 计数到 0 时便停止。

初始时，计数器 1 预置的是与 −55℃相对应的一个预置值。以后计数器 1 每一个循环的预置数都由斜率累加器提供。为了补偿振荡器温度特性的非线性性，斜率累加器提供的预置数也随温度相应变化。计数器 1 的预置数也就是在给定温度处使温度寄存器寄存值增加 1℃计数器所需要的计数个数。

DS18B20 内部的比较器以四舍五入的量化方式确定温度寄存器的最低有效位。在计数器 2 停止计数后，比较器将计数器 1 中的计数剩余值转换为温度值后与 0.25℃进行比较，若低于 0.25℃，温度寄存器的最低位就置 0；若高于 0.25℃，最低位就置 1；若高于 0.75℃时，温度寄存器的最低位就进位然后置 0。这样，经过比较后所得的温度寄存器的值就是最终读取的温度值了，其最后位代表 0.5℃，四舍五入最大量化误差为 ±1/2LSB，即 0.25℃。

温度寄存器中的温度值以 9 位数据格式表示，最高位为符号位，其余 8 位以二进制补码形式表示温度值。测温结束时，这 9 位数据转存到暂存存储器的前两个字节中，符号位占用第 1 字节，8 位温度数据占据第 2 字节。

DS18B20 测量温度时使用特有的温度测量技术。DS18B20 内部的低温度系数振荡器能产生稳定的频率信号；同样的，高温度系数振荡器则将被测温度转换成频率信号。当计数门打开时，DS18B20 进行计数，计数门开通时间由高温度系数振荡器决定。芯片内部还有斜率累加器，可对频率的非线性度加以补偿。测量结果存入温度寄存器中。一般情况下的温度值应该为 9 位，但因符号位扩展成高 8 位，所以最后以 16 位补码形式读出。

DS18B20 工作过程一般遵循以下协议：初始化——ROM 操作命令——存储器操作命令——处理数据

1. 初始化

单总线上的所有处理均从初始化序列开始。初始化序列包括总线主机发出一复位脉冲，接着由从器件送出存在脉冲。存在脉冲让总线控制器知道 DS1820 在总线上且已准备好操作。

2. ROM 操作命令

一旦总线主机检测到从器件的存在，它便可以发出器件 ROM 操作命令之一。所有 ROM 操作命令均为 8 位长。这些命令如下：

1）Read ROM（读 ROM）［33h］：此命令允许总线主机读 DS18B20 的 8 位产品系列编码，唯一的 48 位序列号，以及 8 位的 CRC。此命令只能在总线上仅有一个 DS18B20 的情况下可以使用。如果总线上存在多于一个的从器件，那么当所有从器件企图同时发送时将发生数据冲突的现象（漏极开路会产生线与的结果）。

2）Match ROM（符合 ROM）［55h］：此命令后继以 64 位的 ROM 数据序列，允许总线主机对多点总线上特定的 DS18B20 寻址。只有与 64 位 ROM 序列严格相符的 DS18B20 才能对后继的存储器操作命令作出响应。所有与 64 位 ROM 序列不符的从器件将等待复位脉冲。此命令在总线上有单个或多个器件的情况下均可使用。

3）Skip ROM（跳过 ROM）［CCh］：在单点总线系统中，此命令通过允许总线主机不提供 64 位 ROM 编码而访问存储器操作来节省时间。如果在总线上存在多于一个的从器件而且在 Skip ROM 命令之后发出读命令，那么由于多个从器件同时发送数据，会在总线上发生数据冲突（漏极开路会产生线与的效果）。

4）Search ROM（搜索 ROM）［F0h］：当系统开始工作时，总线主机可能不知道单线总线上的器件个数或者不知道其 64 位 ROM 编码。搜索 ROM 命令允许总线控制器用排除法识别总线上的所有从器件的 64 位编码。

5）Alarm Search（告警搜索）［ECh］：此命令的流程与搜索 ROM 命令相同。但是，仅

在最近一次温度测量出现告警的情况下，DS18B20 才对此命令作出响应。告警的条件定义为温度高于 TH 或低于 TL。只要 DS18B20 一上电，告警条件就保持在设置状态，直到另一次温度测量显示出非告警值或者改变 TH 或 TL 的设置，使得测量值再一次位于允许的范围之内。存储在 EEPROM 内的触发器值用于告警。

3. 存储器操作命令

1）Write Scratchpad（写暂存存储器）［4Eh］：这个命令向 DS18B20 的暂存器中写入数据，开始位置在地址 2。接下来写入的两个字节将被存到暂存器中的地址位置 2 和 3。可以在任何时刻发出复位命令来中止写入。

2）Read Scratchpad（读暂存存储器）［BEh］：这个命令读取暂存器的内容。读取将从字节 0 开始，一直进行下去，直到第 9（字节 8，CRC）字节读完。如果不想读完所有字节，控制器可以在任何时间发出复位命令来中止读取。

3）Copy Scratchpad（复制暂存存储器）［48h］：这条命令把暂存器的内容复制到 DS18B20 的 E^2 存储器里，即把温度报警触发字节存入非易失性存储器里。如果总线控制器在这条命令之后跟着发出读时间隙，而 DS18B20 又正在忙于把暂存器复制到 E^2 存储器，DS18B20 就会输出一个“0”，如果复制结束，DS18B20 则输出“1”。如果使用寄生电源，总线控制器必须在这条命令发出后立即起动强上拉并最少保持 10ms。

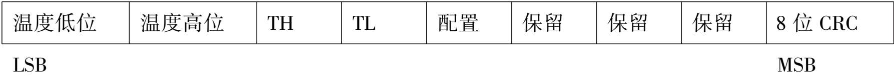
4）Convert T（温度变换）［44h］：这条命令启动一次温度转换而无需其他数据。温度转换命令被执行，而后 DS18B20 保持等待状态。如果总线控制器在这条命令之后跟着发出读时间隙，而 DS18B20 又忙于做时间转换的话，DS18B20 将在总线上输出“0”，若温度转换完成，则输出“1”。如果使用寄生电源，总线控制器必须在发出这条命令后立即起动强上拉，并保持 500ms。

5）Recall E^2（重新调整 E^2）［B8h］：这条命令把存储在 E^2 中温度触发器的值重新调至暂存存储器。这种重新调出的操作在对 DS18B20 上电时也自动发生，因此只要器件一上电，暂存存储器内就有了有效的数据。在这条命令发出之后，对于所发出的第一个读数据时间片，器件会输出温度转换忙的标识：“0”为忙，“1”为准备就绪。

6）Read Power Supply（读电源）［B4h］：对于在此命令发送至 DS18B20 之后所发出的第一读数据的时间片，器件都会给出其电源方式的信号：“0”为寄生电源供电，“1”为外部电源供电。

4. 处理数据

DS18B20 的高速暂存存储器由 9 个字节组成，其分配如图 10-45 所示。当温度转换命令发布后，经转换所得的温度值以 2 字节补码形式存放在高速暂存存储器的第 0 和第 1 个字节。单片机可通过单线接口读到该数据，读取时低位在前，高位在后。



温度低位	温度高位	TH	TL	配置	保留	保留	保留	8 位 CRC

LSB MSB

图 10-45　字节分配

表 10-12 所列是 DS18B20 温度采集转化后得到的 12 位数据，存储在 DS18B20 的两个 8bit 的 RAM 中，二进制中的前面 5 位是符号位，如果测得的温度大于或等于 0，这 5 位为 0，只要将测到的数值乘以 0.0625 即可得到实际温度；如果温度小于 0，这 5 位为 1，测到

的数值需要取反加 1 再乘以 0.0625 即可得到实际温度。

表 10-12　DS18B20 温度数据表

温度/℃	二进制表示												十六进制表示
	符号位（5 位）	数据位（11 位）											
+125	0 0 0 0 0	1	1	1	1	1	0	1	0	0	0	0	07D0H
+25.0625	0 0 0 0 0	0	0	1	1	0	0	1	0	0	0	1	0191H
+10.125	0 0 0 0 0	0	0	0	1	0	1	0	0	0	1	0	00A2H
+0.5	0 0 0 0 0	0	0	0	0	0	0	0	1	0	0	0	0008H
0	0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0000H
-0.5	1 1 1 1 1	1	1	1	1	1	1	1	1	0	0	0	FFF8H
-10.125	1 1 1 1 1	1	1	1	0	1	0	1	1	1	1	0	FF5EH
-25.625	1 1 1 1 1	1	1	0	0	1	1	0	1	1	1	1	FE6FH
-55	1 1 1 1 1	1	0	0	1	0	0	1	0	0	0	0	FC90H

温度转换计算方法举例：

例如当 DS18B20 采集到 +125℃的实际温度后，输出为 07D0H，则：

实际温度 = 07D0H × 0.0625 = 2000 × 0.0625 = 125℃。

例如当 DS18B20 采集到 -55℃的实际温度后，输出为 FC90H，则应先将 11 位数据位取反加 1 得 370H（符号位不变，也不作为计算），则：

实际温度 = 370H × 0.0625 = 880 × 0.0625 = 55℃。

5. 时序

主机使用时间隙（time slots）来读写 DS18B20 的数据位和写命令字的位。初始化时序如图 10-46 所示，读写时序如图 10-47 所示。

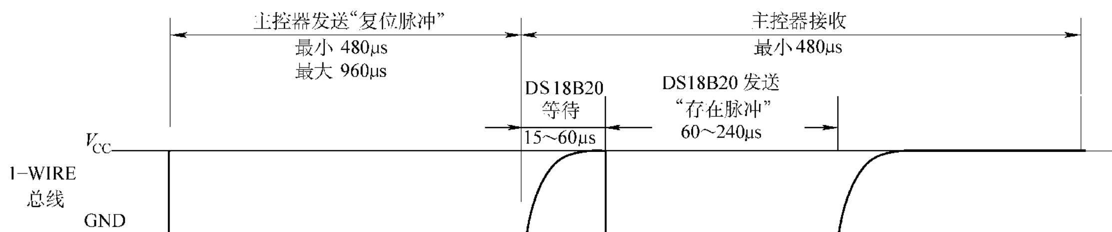


图 10-46　初始化时序

10.6.6　DS18B20 温度测试软、硬件设计

1. 硬件原理

本实例以 DS18B20 作为温度传感器，单片机为 89C51，DS18B20 的 DQ 端与 89C51 的 P33 相连。温度值通过数码管显示。具体硬件原理如图 10-48 所示。

2. 软件功能

该软件的主要功能是使用 89C51 单片机和温度传感器 DS18B20 对温度进行测量和显示。本程序只用于原理性学习，软件功能较少，温度显示也只能显示正温度值。软件流程图如图

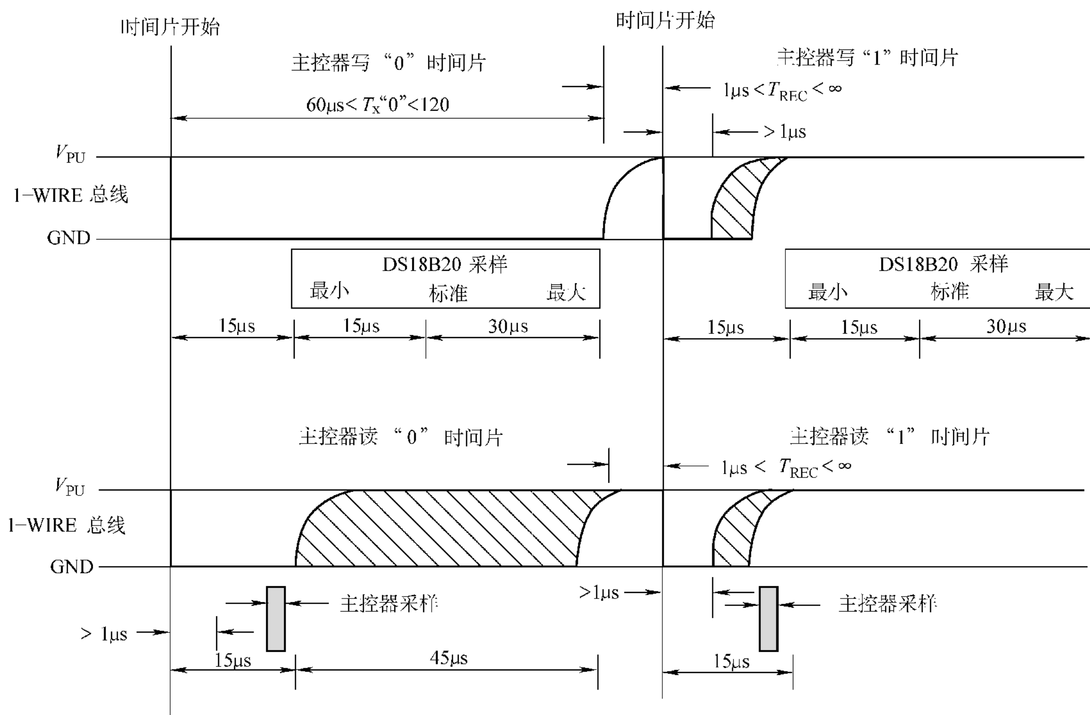


图 10-47　读写时序

10-49 所示。

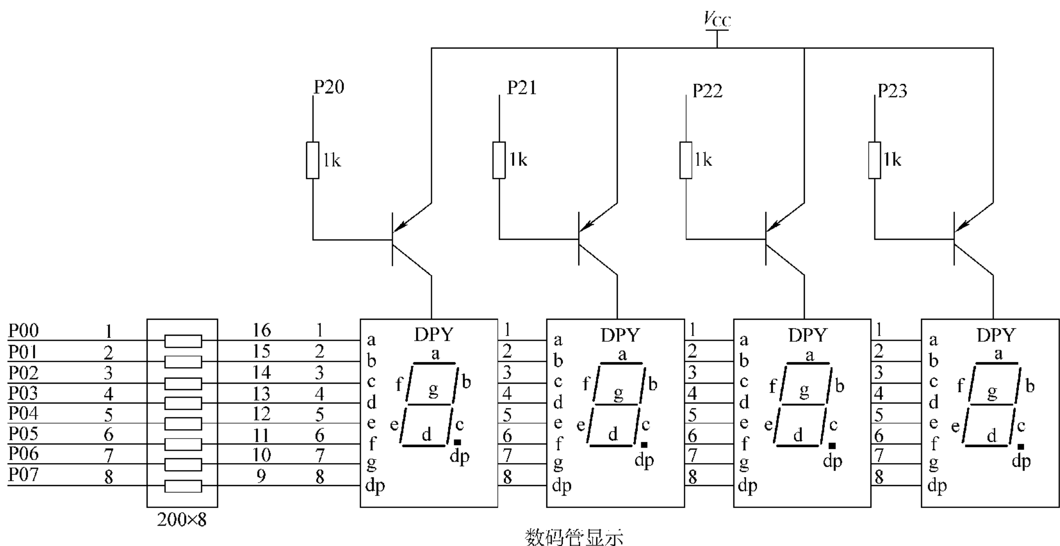
3. 软件代码

```
/*****************************/
/*      杭州电子 & 计算机工作室      */
/*      http://www.hificat.com      */
/*      DS18B20 测温程序      */
/*      目标器件:AT89C51      */
/*      晶振:12MHz      */
/*      编译环境:Keil 7.50A      */
/*****************************/
#include <reg51.h>
#include <absacc.h>

unsigned char code tab[] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

sbit DQ = P1^0;                        //数据传输线接单片机的相应引脚

unsigned char tempL = 0;               //设全局变量
unsigned char tempH = 0;
float temperature;                     //温度值保存在 temperature 里
```



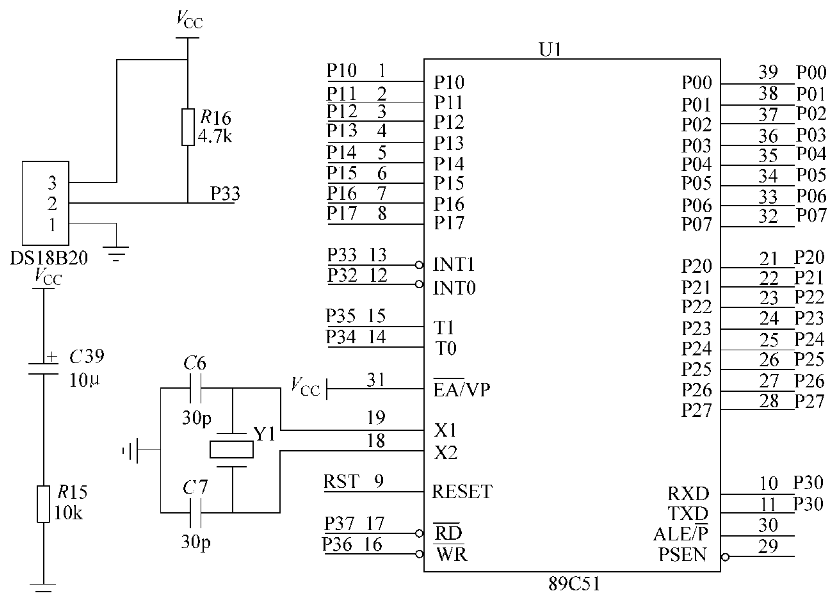


图 10-48　硬件原理

```
/*********************************************
函数功能:延时子程序
入口参数:time
出口参数:
*********************************************/
void delay(unsigned int time)
{
    unsigned int n;
```

```
    n = 0;
    while(n < time)
    {n++;}
    return;
}
/****************************************
函数功能:数码管扫描延时子程序
入口参数:
出口参数:
****************************************/
void delay1(void)
{
    int k;
    for(k = 0;k < 600;k++);
}
```

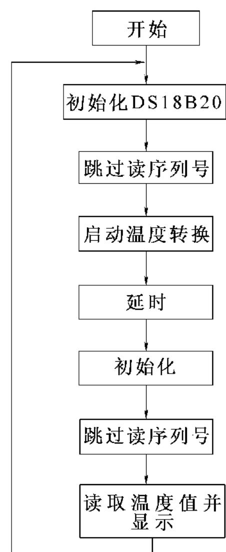


图 10-49　软件流程图

```
/********************************************
函数功能:数码管显示子程序
入口参数:k
出口参数:
********************************************/
void display(int k)
{
    P2 = 0xfe;
    P0 = tab[k/1000];
    delay1();
    P2 = 0xfd;
    P0 = tab[k%1000/100];
    delay1();
    P2 = 0xfb;
    P0 = tab[k%100/10];
    delay1();
    P2 = 0xf7;
    P0 = tab[k%10];
    delay1();
    P2 = 0xff;
}
/********************************************
函数功能:DS18B20 初始化子程序
入口参数:
```

出口参数：

```
****************************************/
Init_DS18B20(void)
{
    unsigned char x=0;
    DQ=1;                                  //DQ 先置高
    delay(8);                              //稍延时
    DQ=0;                                  //发送复位脉冲
    delay(85);                             //延时(>480ms)
    DQ=1;                                  //拉高数据线
    delay(14);                             //等待(15~60ms)
    x=DQ;                                  //用 X 的值来判断初始化有没有成功,
                                             18B20 存在的话 X=0,否则 X=1
    delay(20);
}
/****************************************
函数功能:向 DS18B20 读一字节数据
入口参数:
出口参数:dat
****************************************/
ReadOneChar(void)                          //主机数据线先从高拉至低电平 1ms 以
                                             上,再使数据线升为高电平,从而产生
                                             读信号
{
    unsigned char i=0;                     //每个读周期最短的持续时间为 60ms,
                                             各个读周期之间必须有 1ms 以上的高
                                             电平恢复期
    unsigned char dat=0;
    for (i=8;i>0;i--)                      //一个字节有 8 位
        {
            DQ=1;
        delay(1);
        DQ=0;
        dat>>=1;
            DQ=1;
        if(DQ)
            dat|=0x80;
        delay(4);
        }
```

```
    return(dat);
}
/****************************************
函数功能:向 DS18B20 写一字节数据
入口参数:dat
出口参数:
****************************************/
WriteOneChar(unsigned char dat)
{
    unsigned char i=0;                  //数据线从高电平拉至低电平,产生写起
                                          始信号。
   for(i=8;i>0,i--)
    {
        DQ=0;
        DQ=dat&0x01;
        delay(5);
        DQ=1;
        dat>>=1;
    }
        delay(4);
}
/****************************************
函数功能:向 DS18B20 读温度值
入口参数:
出口参数:temperature
****************************************/
ReadTemperature(void)
{
    Init_DS18B20();                     //初始化
    WriteOneChar(0xcc);                 //跳过读序列号的操作
    WriteOneChar(0x44);                 //启动温度转换
    delay(125);                         //转换需要一点时间,延时
    Init_DS18B20();                     //初始化
    WriteOneChar(0xcc);                 //跳过读序列号的操作
    WriteOneChar(0xbe);                 //读温度寄存器(头两个值分别为温度的
                                          低位和高位)
    tempL=ReadOneChar();                //读出温度的低位 LSB
    tempH=ReadOneChar();                //读出温度的高位 MSB
    temperature=((tempH*256)+tempL)*0.0625; //温度转换,把高低位做相应的运算转化
```

```
                                        为实际温度
    delay(200);
    return(temperature);
}
/*********************************************
函数功能:主程序
入口参数:
出口参数:
*********************************************/
void main()
{
    float i;
    while(1)
    {
        i = ReadTemperature();
        display(i);
    }
}
```

10.7 无线通信模块应用

在一些特殊场合，系统间的通信不能采用传统的有线通信方式而是需要无线数据传输的方式。目前，市场上出现了很多无线数据收发模块，如 PTR 2000、FB230 等。无线数据收发模块的性能优异，外围元器件少，设计、使用非常简单。无线收发模块一般在内部都集成了高频发射、高频接收、PLL 合成、FSK 调制/解调、参数放大、功率放大等功能。在无线



图 10-50　无线收发模块实物图

遥控领域，PT2262/PT2272 是目前最常用的芯片之一，本节就重点介绍这两个芯片的原理及相应模块的应用。无线收发模块的实物如图 10-50 所示。

10.7.1 PT2262/PT2272 编码/解码芯片原理简介

PT2262/PT2272 是中国台湾普城公司生产的一种 CMOS 工艺制造的低功耗低价位通用编码/解码电路，是目前在无线通信电路中作地址编码识别最常用的芯片之一。PT2262/PT2272 最多可有 12 位（A0～A11）三态（悬空，接高电平，接低电平）地址设定引脚，任意组合可提供 531441 个地址码。PT2262 最多可有 6 位（D0～D5）数据端引脚，设定的地址码和数据码从引脚 17（Dout）串行输出，可用于无线遥控发射电路。

PT2262 和 PT2272 的引脚排列如图 10-51 所示。对于编码器 PT2262，A0～A5 共 6 根线为地址线，而 A6～A11 共 6 根线可以作为地址线，也可以作为数据线，这要取决于所配合使用的解码器。若解码器没有数据线，则 A6～A11 作为地址线使用，这种情况下，A0～A11 共 12 根地址线，每线都可以设置成“1”、“0”、“开路”三种状态之一，因此共有编码数 $3^{12}=531441$ 种；但若配对使用的解码器的 A6～A11 是数据线，例如 PT2272，那么这时 PT2262 的 A6～A11 也作为数据线用，并只可设置为“1”和“0”两种状态之一，而地址线只剩下 A0～A5 共 6 根，编码数降为 $3^{6}=729$ 种。

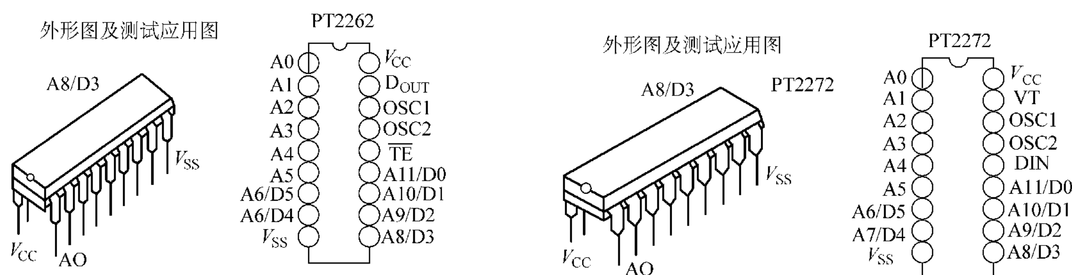


图 10-51 PT2262/PT2272 的引脚排列

该编码解码器的编码信号格式是：用 2 个周期的占空比为 1∶3（即高电平宽度为 1，低电平宽度为 2，周期为 3）的波形来表示 1 个“0”，用 2 个周期的占空比为 2∶3（即高电平宽度为 2，低电平宽度为 1，周期为 3）的波形来表示 1 个“1”，用 1 个周期的占空比为1∶3 的波形紧跟着 1 个周期的占空比为 2∶3 的波形来表示“开路”。地址码和数据码都用宽度不同的脉冲来表示，两个窄脉冲表示“0”；两个宽脉冲表示“1”；一个窄脉冲和一个宽脉冲表示“F”也就是地址码的“悬空”。

编码芯片 PT2262 发出的编码信号由地址码、数据码、同步码组成一个完整的码字。解码芯片 PT2272 接收到信号后，其地址码经过两次比较核对后，VT 端才输出高电平，与此同时，相应的数据端也输出高电平。PT2262 每次至少发射 4 组字码，因为无线发射的特点，第一组字码非常容易受零电平干扰，往往会产生误码，所以 PT2272 只有在连续两次检测到相同的地址码加数据码才会把数据码中的“1”驱动相应的数据输出端为高电平和驱动 VT 端同步为高电平。当发射机没有按键按下时，PT2262 不接通电源，其引脚 17 为低电平，所以 315MHz 的高频发射电路不工作。当有按键按下时，PT2262 得电工作，其引脚 17 输出经调制的串行数据信号，当引脚 17 为高电平期间 315MHz 的高频发射电路起振并发射等幅高频信号，当引脚 17 为低平期间 315MHz 的高频发射电路停止振荡，所以高频发射电路完全

收控于 PT2262 的引脚 17 输出的数字信号，从而对高频电路完成幅度键控（ASK 调制）相当于调制度为 100% 的调幅。

PT2272 解码芯片有不同的后缀，表示不同的功能，有 L4/M4/L6/M6 之分。其中 L 表示锁存输出，数据只要成功接收就能一直保持对应的电平状态，直到下次遥控数据发生变化时改变。M 表示非锁存输出，数据端输出的电平是瞬时的而且和发射端是否发射相对应，可以用于类似点动的控制。后缀的 6 和 4 表示有几路并行的控制通道，当采用 4 路并行数据时（PT2272-M4），对应的地址编码是 8 位，如果采用 6 路的并行数据时（PT2272-M6），对应的地址编码是 6 位。

PT2262 和 PT2272 除地址编码必须完全一致外，振荡电阻也必须匹配，一般要求解码器振荡频率要高于编码器振荡频率的 2.5 ~ 8 倍，否则接收距离会变近甚至无法接收，随着技术的发展，市场上出现一批兼容芯片，在实际使用中只要对振荡电阻稍做改动就能配套使用。在具体的应用中，外接振荡电阻可根据需要进行适当的调节，阻值越大振荡频率越低，编码的宽度越大，发码一帧的时间越长。市场上大部分产品都是用 2262/1.2M = 2272/200K 组合的，少量产品用 2262/4.7M = 2272/820K。

PT2262 编码电路与 PT2272 解码电路一般配对使用，PT2262 的特点是在其内部已经把编码信号调制在了一个较高的载频上。要把遥控编码信息用无线方式（红外线或无线电等）传送出去，必须有载体（载波），把编码信息“装载”在载体上（调制在载波上）才能传送出去，因此需要一个振荡电路和一个调制电路。PT2262 编码器内部，已包含了这些电路，从 D_{OUT} 端送出的是调制好了的约 38kHz 的高频已调波，因此使用起来非常方便，适用于红外线和超声波遥控电路。PT2262/PT2272 的引脚说明分别见表 10-13 和表 10-14。

表 10-13　编码电路 PT2262 的引脚说明

符号	引脚号	说　　明
D0 ~ D5	7 ~ 8、10 ~ 13	数据输入端，有一个为“1”即有编码发出，内部下拉
V_{cc}	18	电源正端（+）
V_{ss}	9	电源负端（−）
TE	14	编码启动端，用于多数据的编码发射，低电平有效
0SC1	16	振荡电阻输入端，与 0SC2 所接电阻决定振荡频率
0SC2	15	振荡电阻、振荡器输出端
D_{OUT}	17	编码输出端（正常时为低电平）

表 10-14　解码电路 PT2272 的引脚说明

符号	引脚号	说　　明
A0 ~ A11	1 ~ 8、10 ~ 13	地址引脚，用于进行地址编码，可置为“0”，“1”，“f”（悬空），必须与 PT2262 一致，否则不解码
D0 ~ D5	7 ~ 8、10 ~ 13	地址或数据引脚，当作为数据引脚时，只有在地址码与 PT2262 一致，数据引脚才能输出与 PT2262 数据端对应的高电平，否则输出为低电平，锁存型只有在接收到下一数据才能转换
V_{cc}	18	电源正端（+）
V_{ss}	9	电源负端（−）
DIN	14	数据信号输入端，来自接收模块输出端
0SC1	16	振荡电阻输入端，与 0SC2 所接电阻决定振荡频率
0SC2	15	振荡电阻、振荡器输出端
VT	17	解码有效确认，输出端（常低）解码有效变成高电平（瞬态）

10.7.2 编码发射模块简介

编码发射模块外形小巧、美观，与很多车辆防盗系统中的遥控器一样。根据功能的多少按键数也不一样，我们本节所用的发射模块为 A、B、C、D 四个按键。编码发射模块主要由 PT2262 编码 IC 和高频调制、功率放大电路组成，常用的编码发射模块实物和内部框图如图 10-52 所示。



图 10-52　编码发射模块实物与原理框图

其中编码部分电路由 PT2262 编码 IC 来组成，具体电路如图 10-53 所示。

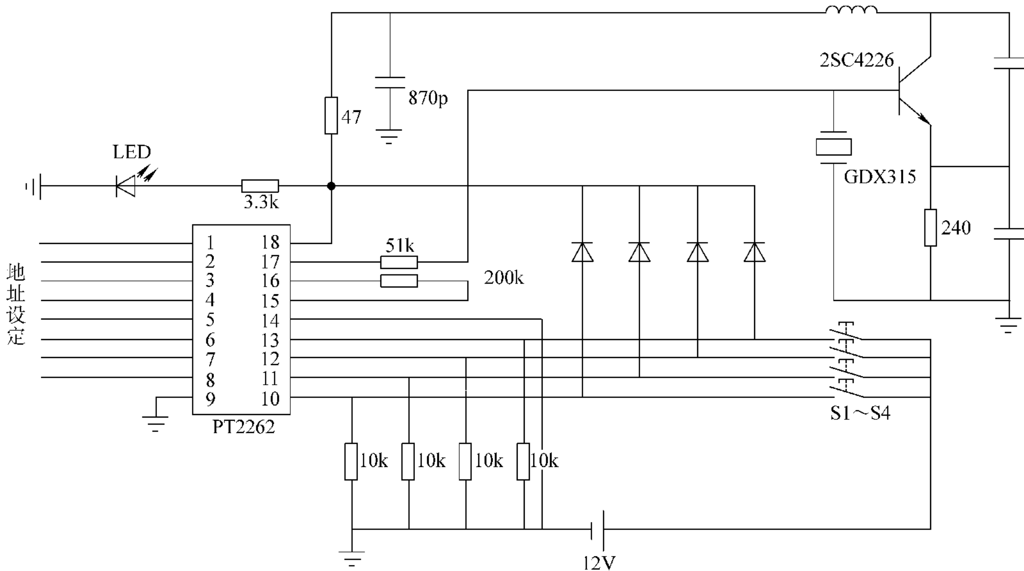


图 10-53　编码电路原理

10.7.3 解码接收模块

解码接收模块包括接收头和解码芯片 PT2272 两部分组成。接收头将收到的信号输入 PT2272 的引脚 14（DIN），PT2272 再将收到的信号解码。解码接收模块如图 10-54 和图 10-

55 所示，解码接收模块电路原理如图 10-56 所示。

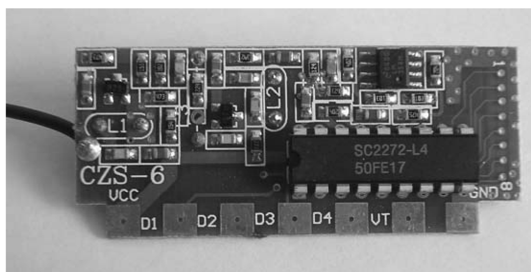


图 10-54　无线接收模块正面

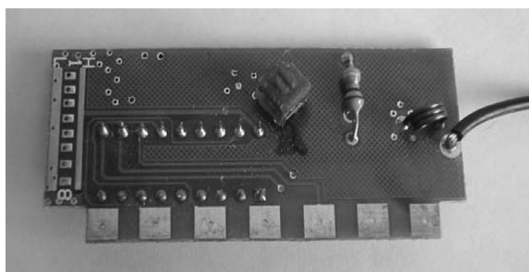


图 10-55　无线接收模块反面

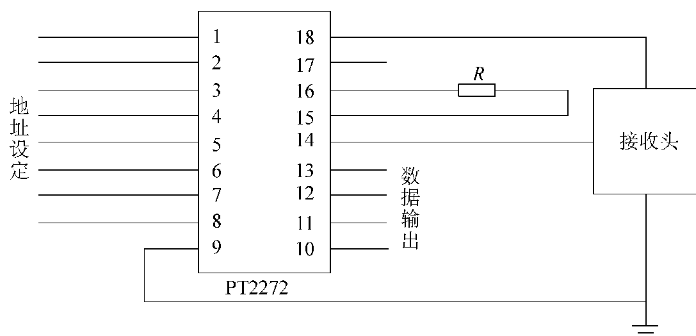


图 10-56　解码接收模块的电路原理

10.7.4　PT2262/PT2272 芯片的地址编码设定

在通常使用中，我们一般采用 8 位地址码和 4 位数据码，这时编码芯片 PT2262 和解码芯片 PT2272 的引脚 1 ~ 8 为地址设定端，有三种状态可供选择：悬空、接正电源、接地三种状态，地址编码不重复度为 $3^8=6561$ 组，只有发射端 PT2262 和接收端 PT2272 的地址编码完全相同，才能配对使用。遥控模块的生产厂家为了便于生产管理，出厂时遥控模块的 PT2262 和 PT2272 的八位地址编码端全部悬空，这样用户可以很方便地选择各种编码状态。用户如果想改变地址编码，只要将 PT2262 和 PT2272 的引脚 1 ~ 8 设置相同即可，例如将发射机的 PT2262 的引脚 2 接地，引脚 3 接正电源，其他引脚悬空，那么接收机的 PT2272 只要也将引脚 2 接地，引脚 3 接正电源，其他引脚悬空就能实现配对接收。地址设置跳线如图 10-57 所示，用户可以在 PCB 板上直接将地址引脚（PCB 板中间 8 个过孔焊盘）与 L（低电平）或 H（高电平）相连，从而实现地址设置。PT2262 与 PT2272 地址设置要完全一样。当两者地址编码完全一致时，接收机对应的 D1 ~ D4 端输出约 4V 互锁高电平控制信号，同时 VT 端也输出解码有效高电平信号。

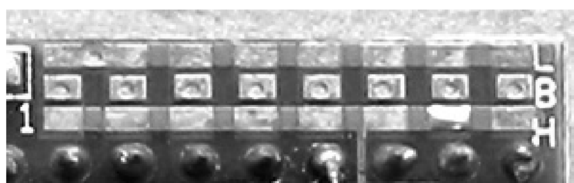


图 10-57　地址设置跳线图

10.7.5　基于单片机的无线收发模块应用

在功能稍复杂的系统中仅靠一对无线收发模块往往达不到要求，很多情况下都要借助于单

片机扩展出更多的功能。本例通过一个简单的例子，实现单片机与无线接收模块的组合应用。

1. 实例功能

在发射模块上按下 A、B、C、D 四个键，接收模块将接收到的数据传送给单片机，在单片机上实现 LED 数码管显示。A、B、C、D 分别对应 1、2、3、4。即在发射模块上按下 A 按键，对应单片机接收后在 LED 数码管上显示 0001，按下 B 键显示 0002……。

2. 硬件原理

无线接收模块的数据输出端与 89C51 的 P32 ~ P35 相连，VT 端通过一个晶体管反相连到 P37 用于查询状态，4 个 LED 数码管用于显示数字。实验设备如图 10-58 所示，硬件原理如图 10-59 所示。

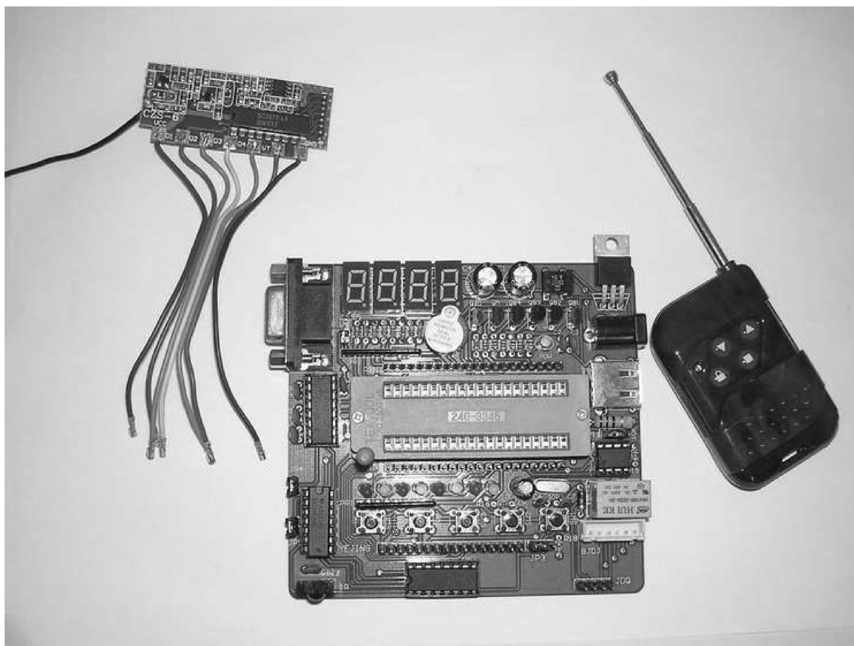


图 10-58　用于无线遥控实验的实验板、遥控器、接收解码模块

3. 软件代码

```
/*****************************************/
/*          杭州电子 & 计算机工作室          */
/*          http://www.hificat.com         */
/*          无线收发演示程序                 */
/*          目标器件：AT89C51               */
/*          晶振：12MHz                     */
/*          编译环境：Keil 7.50A            */
/*****************************************/

/********************包含头文件*******************/
#include <reg51.h>
```

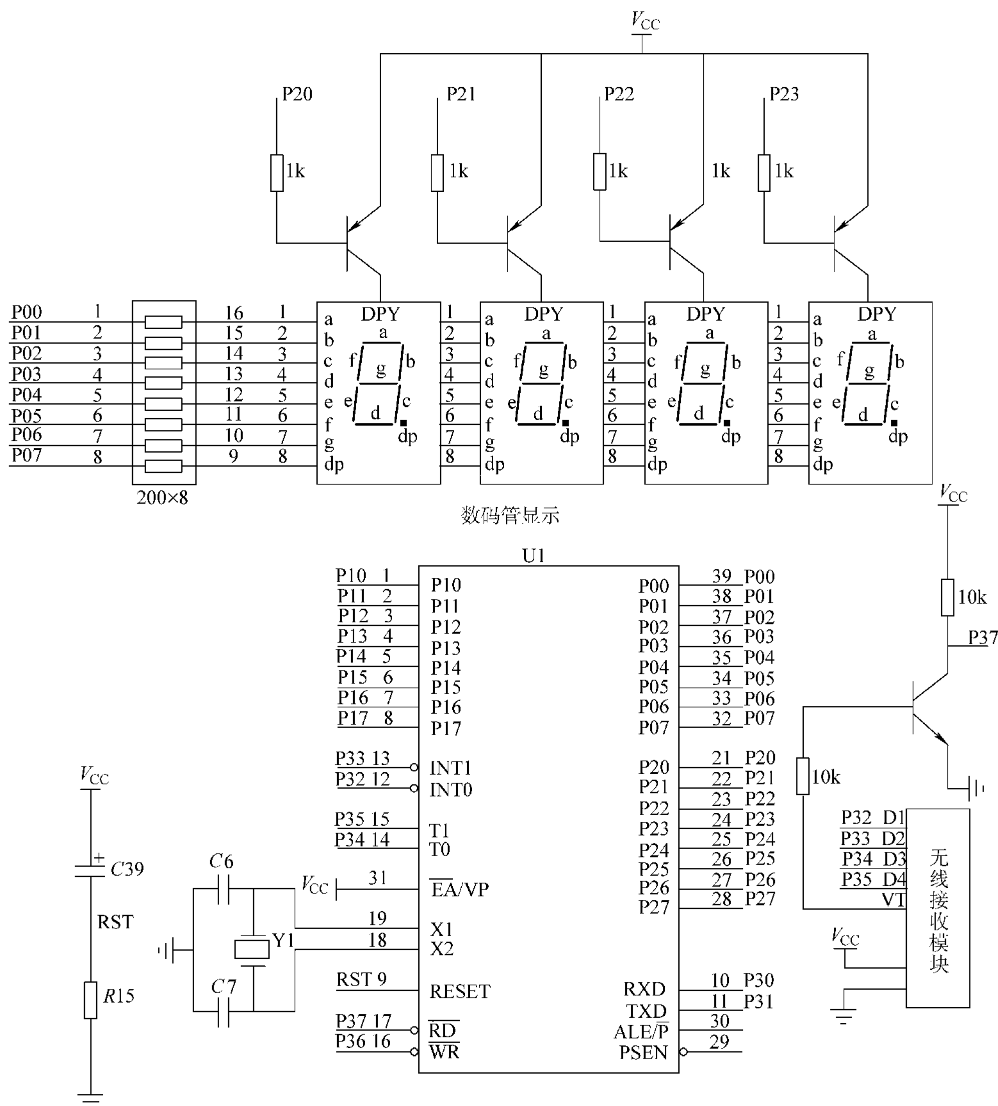


图 10-59　硬件原理

```
/*******************共阳LED段码表*****************/
unsigned char code tab[ ] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
/*******************定义全局变量******************/

/*********************端口定义********************/
char dat;                           //接收到的数据
sfr dport  =  0xB0;                 //定义数据口为 P3 口
sbit VT  =  P3^7;                   //VT 端定义为 P37
/*****************************************************
函数功能：LED 数码管显示程序
```

```
入口参数：k
出口参数：
*****************************************/
void display (int k)
    {
    P2 =0xfe;                   //位选
    P0 = tab [k/1000];          //显示千位数字
    delay ();                   //延时
    P2 =0xfd;                   //位选
    P0 = tab [k%1000/100];      //显示百位数字
    delay ();                   //延时
    P2 =0xfb;                   //位选
    P0 = tab [k%100/10];        //显示十位数字
    delay ();                   //延时
    P2 =0xf7;                   //位选
    P0 = tab [k%10];            //显示个位数字
    delay ();                   //延时
    P2 =0xff;                   //位选
}

/*****************************************
函数功能：主程序
入口参数：
出口参数：
*****************************************/
void main (void)
    {
    char datavalue;
    P2 =0xff;                   //端口初始化，关 LED 显示
    P0 =0xff;
    VT =1;                      //置输入状态
    while (1)
    {
        if (VT = =0)
          {
        dat = (dport&0x0F);     //读取 P3 口低四位数据
        }
        else
          {
```

```
        datavalue = 0;
        }
        if (dat = = 0x01)
        datavalue = 0x01;
        if (dat = = 0x02)
        datavalue = 0x02;
        if (dat = = 0x04)
        datavalue = 0x03;
        if (dat = = 0x08)
        datavalue = 0x04;
        display (datavalue);        //将读到的数显示在数码管
    }
}
```

10.8 X25045/X5045 多功能器件的应用

X25045 是将可编程看门狗、电压监控、E^2PROM 集成于一体的多功能芯片。该芯片是美国 xicor 公司的新型产品，具有体积小、占用 I/O 少等优点，应用于系统中可以简化单片机系统的设计，并完善其性能，而 X5045 是 X25045 后来的型号，功能同 X25045 完全一样。在这节里，我们就来看看为什么我们需要在系统中使用它，以及如何来使用它。

10.8.1 看门狗、电压监控概述

由于已经学习过 E^2PROM AT24C01 的使用，所以在这里不打算介绍 X25045 E^2PROM 部分的使用，我们的重点是学习看门狗和电压监控技术。

1. 看门狗

看门狗（Watchdog）电路是嵌入式系统需要的抗干扰措施之一。

记得刚开始学习单片机的时候，有个很天真的想法，就是觉得只要我的程序写对了，只要能够正确的运行，那设计过程也就完成了，然后当客户说产品不好用的时候，我还会很疑惑的说：“不可能吧？”

这样的想法，是因为我们有一个假设，就是我们的程序永远能够按照我们的意图顺利执行，单片机本身也不会因为任何原因而产生不正常的执行过程。不过事实上呢？或许在实验室里，一个系统跑上几个月都没有问题，但是一到现场，就有可能不时地出现问题，这样的事我自己是遇到过的。比如一个系统，运行着运行着就死机了，断电后来重新运行则又正常了。原因何在？第一个可能，也是最可能的地方，就是程序设计的时候，存在疏漏。当程序大了之后，要考虑到所有的细节，不是说不可能，但是如果时间很有限，就变得很难了，而且一般测试验证过程也无法保证经历所有的可能。所以极有可能，在某个极为特别的情况下，系统会死机；第二个可能，系统的本身受到干扰，包括人体静电等很多因素都可能导致系统复位，甚至死机，这或者应该归咎于系统设计的问题，也有可能是采用的单片机本身就

无法抵抗这个程度的干扰。因此，在单片机系统的发展过程中，产生了不少专门为提高系统可靠性的技术，此中有软件的，也有硬件的。而硬件看门狗技术无疑是其中很出色的一种，现在绝大部分系统里都会带有看门狗，而且现在很多 MCU 本身就已经内嵌了它。

乍听起来，很奇妙是不是？不过这个电路的确可以起到这样的作用。我们先看看它是如何工作的。我们先来想想一个情况，就是上面说到的，现在一个系统突然死了，我们想让他活过来，怎么办？断电然后上电？还有么？有，复位。是的，复位也是可以的，因为我们没有足够的时间对所有的可能进行验证也不可能预料到所有的干扰而添加十分有效的防范措施。所以我们希望系统可以这样做：万一系统真的死机了，要能在足够短的时间内重新复位并开始工作。“足够短”的程度要视情况而定，所以我们需要一个能够定时复位的电路，比如每隔 1 秒钟，这个电路会自动把系统复位。这样的电路在以前的系统中的确是有使用的，但是大部分时候，当系统在正常运行的时候，我们并不希望系统复位，所以这个定时复位电路还需要有另外一个功能，就是可以从外部把计时清零，这样一来，因为计数器没有计数溢出，就不会产生复位信号，就可是使系统按照正常的执行过程执行。能够实现上面所描述的功能的电路就叫做“看门狗”，而外部清零则叫做“喂狗”，这只是一个形象的比喻而已。

2. 电压监控

通常说电压监控的时候，还会说到另外一点，就是“门限电压”，也就是当供电电压一旦高于这个数值时，复为过程就允许产生作用，否则复位信号一直使复位端处于复位状态，同样，掉电的时候也有这样的过程，为什么需要呢？

原来当单片机的工作电压低于某个数值的时候，它的很多状态具有不确定性，而很多不确定的操作有可能导致操作上的错误，数据被更改甚至是丢失，因此也就有了电压监控电路，来保证系统能够在上下电的时候能够安全进入和退出工作状态。

10.8.2 X25045/X5045 的结构及工作原理

X25045/X5045 的引脚排列如图 10-60 所示。

1. X25045/X5045 的结构

X25045/X5045 的结构如图 10-61 所示，主要功能组成部分为：看门狗电路、存储器电路、上下电复位电路。其中看门狗电路包含喂狗信号输入电路、看门狗定时器、复位信号输出等部分；E^2PROM 部分则包含了数据寄存器、命令寄存器、数据保护逻辑、E^2PROM 阵列、状态寄存器等；上下电检测则包含带隙基准源、比较器、复位信号产生电路等。具体的逻辑关系可以从图 10-61 中看的一目了然。

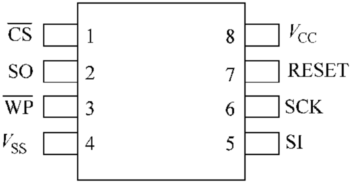


图 10-60 X25045/X5045 的外部引脚排列

$\overline{CS}$—芯片选择输入 SO—串行数据输出 $\overline{WP}$—写保护 V_{ss}—地 SI—串行数据输入 SCK—串行时钟输入 RESET—复位输出，高有效 V_{cc}—电源电压

2. X25045/X5045 的 E^2PROM

把看门狗电路和 E^2PROM 结合起来，是 X25045/X5045 的一大特点。通过 SPI 总线，可以访问芯片内部的 512 字节串行 E^2PROM，它们是采用 CMOS 工艺制造的，每个字节可擦写 10 万次以上，并可保存数据 100 年以上。其接口时序如图 10-62 所示。

芯片工作期间，$\overline{CS}$端应始终保持低电平。在一个读时序周期内，数据在串行时钟 SCK

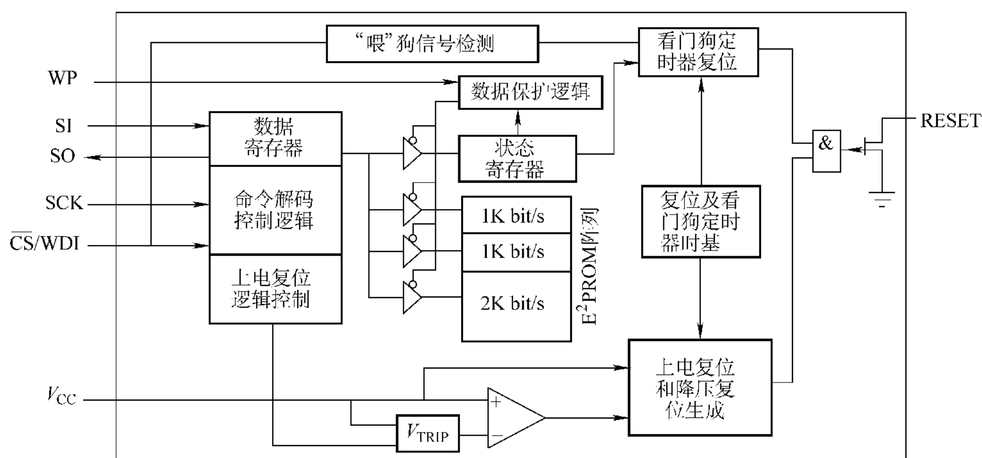


图 10-61　X25045/X5045 的结构

的下降沿，由 SO 端串行输出，而缓冲地址或数据在串行时钟 SCK 的上升沿，由 SI 端输入锁存。当$\overline{CS}$端为低电平时，写保护功能部分可以使用。当$\overline{CS}$端为高电平时，所有写保护功能才正常。值得注意的是，在芯片选通时，$\overline{CS}$端变为低电平，将要中断对 X25045 的写操作，若正处在内部写周期内，则对写操作没有影响。

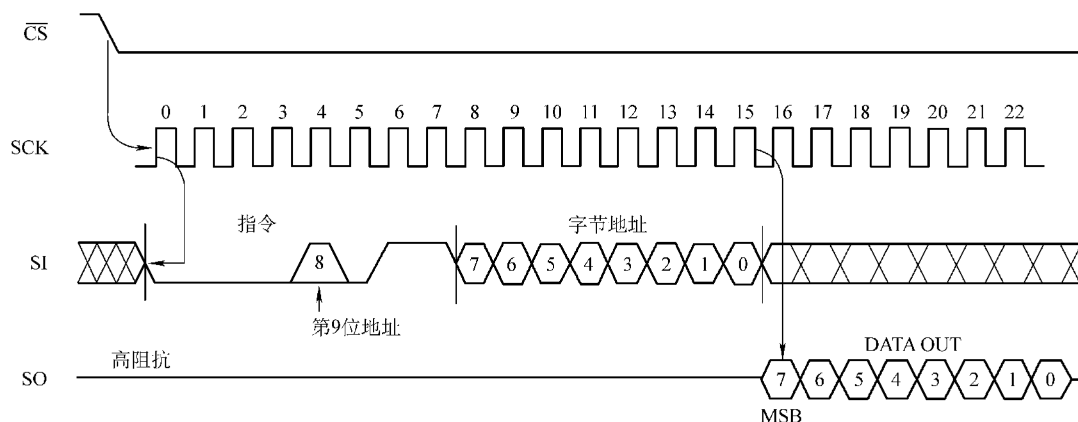


图 10-62　X25045/X5045 内部的 E²PROM 访问时序

对 X25045/X5045 内部的 E²PROM 的访问是通过一个 8bit 指令寄存器来实现的，该寄存器可以通过引脚 SI 来访问，数据在串行时钟输入的上升沿由时钟同步输入。另外，对芯片的所有操作都需要通过对该寄存器的写命令来完成。其中操作主要有：

1）WREN：设置写使能锁存器（允许写）。

2）WRDI：复位写使能锁存器（禁止写）。

3）RDSR：读状态寄存器。

4）WRSR：写状态寄存器。

5）READ：从所选地址开始的存储器中读出数据。

6）WRITE：把数据写入所选地址开始的存储器中。

在执行写操作之前写使能锁存器必须被置位，在写操作完成后该寄存器必须被复位。其

指令格式不再详细介绍，可参考详细的资料。

X25045 有一个状态寄存器，可以用来提供 X25045 的状态信息以及设置块保护和看门狗的超时功能，其格式为：

7	6	5	4	3	2	1	0
—	—	WD1	WD0	BL1	BL0	WEL	WIP

WIP 位表示 X25045 是否在向 E^2PROM 写数据。该位是 1 时，表示正在进行写操作，此时不能向其写数据；反之，则是没有写操作进行，可以向其写数据。WEL 位是写使能锁存器的状态位。可以由指令进行复位和置位操作。写使能锁存器被复位时向其写操作被禁止。

由 WREN 指令可以对状态寄存器中的 BL0，BL1，WD0，WD1 进行设置。BL0 和 BL1 位确定 E^2PROM 的块保护地址范围。WD0 和 WD1 位是看门狗超时功能的设定位，可以设置不同的周期（典型值 1.4 s、600 ms、200 ms）。当 WD0 和 WD1 同时为 1 时，功能被禁止。

10.8.3　X25045/X5045 和单片机之间的软件接口程序设计

X25045/X5045 需要同单片机接口的硬件连接分两个部分，一个部分是看门狗的复位信号，接单片机的 RESET 端，另外的通信口和写保护端接普通 I/O 口即可。由于接口程序设计的本身比较简单，请自行阅读下面的程序了解接口的实现。

X25045/X5045 和单片机之间的接口程序实现

```
#include <reg52.h>
#include <intrins.h>

sbit si = 0xa1;
sbit so = 0xa2;
sbit cs = 0xa3;
sbit sck = 0xa0;

unsigned char buffer [6];

//串行输入子程序
unsigned char X25045Rece ()
{
    unsigned char i;
    unsigned char var =0;
    for (i =0; i <7; i + +)                    // 8 位字节带进位左移
    {
        sck =1;                                // 产生 SCK 脉冲
        sck =0;
        CY = so;                               // SO 移进位位
```

```
        if (CY) var+ =1;
        var< < =1;
    }
    sck =1;
    sck =0;
    CY = so;
    if (CY) var+ =1;                        // 判断是否结束
    return (var);
}

//串行输出子程序
void X25045Send (unsigned char var)
{
    unsigned char i, temp;
    for (i =0; i <8; i+ +)                  // 8 位字节输出
    {
        sck =0;                             // 使 SCK 为低电平
        temp = var&0x80;                    // 选择高位
        if (temp = =0x80) si =1;            // 输出高位
        else si =0;
        sck =1;                             // 使 SCK 为 1
        var< < =1;
    }                                       // 左移 1 位
    si =0;
}

//写数据寄存器子程序
void Xwrite (unsigned char var, add)
{
    sck =0;                                 // 使 SCK 为低电平
    cs =0;                                  // 芯片选择有效
    X25045Send (0x02);                      // WREN 指令
    X25045Send (add);                       // 调输出子程序
    X25045Send (var);                       // 调输出子程序
    sck =0;                                 // 使 SCK 为低电平
    cs =1;                                  // 芯片选择无效
}
//读数据寄存器子程序
unsigned char XBRead (unsigned char add)
```

```
{
    unsigned char i;
    sck =0;                                   // 使 SCK 为低电平
    cs =0;                                    // 芯片选择有效
    X25045Send (0x03);                        // READ 指令
    X25045Send (add);                         // 调输出子程序
    i = X25045Rece ( );                       // 调输入子程序
    sck =0;                                   // 使 SCK 为低电平
    cs =1;                                    // 芯片选择无效
    return (i);
}

//看门狗复位程序
void reset ( )
{
    cs =1;
    cs =0;
    cs =1;
}

//通用延时函数
void delay ( )
{
    unsigned char i, j;
    for (i =0; i <200; i + +)
        for (j =0; j <200; j + +);
}

//主程序功能
//向 0x10 地址写入数据，并在主循环处读回，
//同时在主循环里清看门狗
void main (void)
{
    unsigned char i;
    Xwrite ( 'x', 0x10);

    while (1)
    {
        Delay ( );
```

```
        buffer［0］ =XBRead（0x10）;
        reset（）;
    }
}
```

10.9 红外遥控的软件解码

现在的很多音频、视频设备都采用红外遥控这种方式。因其体积小、功耗低、成本低，并且不影响周边环境，不干扰其他电器设备，是一种在视距内实现近距离无线遥控的理想方法，如图10-63所示。但也正是因为使用太广泛，同时又不能相互干扰，因此就必须采用不同的控制编码，而不同厂家生产的产品也需要采用不同的代码或者编码。所以目前实际使用的编码方法非常多，常见的恐怕也在30种之上，例如最常见的有3010，6121，7461，9012等等，一般现在的一些单功能的编码芯片还是用数字电路实现的，但是多功能的则大多用4或8位单片机实现。在这节我们将要看看如何用单片机来实现对遥控器编码的解码，当然在解码之前，也会把红外遥控相关的基本知识介绍一下。



图10-63　一些红外遥控器实物图

10.9.1 红外遥控概述

1. 红外线

按照人眼能看到的可见光波长从长到短排列，可见光依次为红、橙、黄、绿、青、蓝、紫，其中红光的波长范围为0.62～0.76μm，紫光的波长范围为0.38～0.46μm，而作为红外遥控信号载体的红外线，就是比红光波长还长的光，是人眼看不见的。如图10-64所示的就是这些光的波长分布。

2. 红外遥控信号发生电路

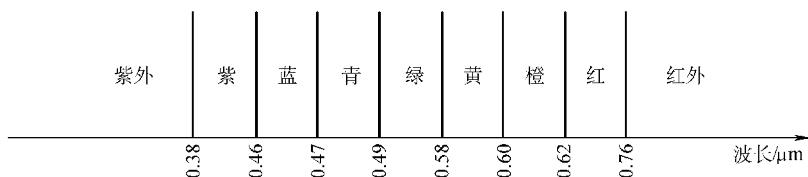


图 10-64　光线波长从短到长的排列

波长大于 0.76μm 的光就是红外线，不过在红外遥控中，通常使用的波长是 0.94μm，而用来产生这种信号的器件则是红外发光二极管。由于使用的材料不同于普通发光二极管，因而在红外发光二极管两端施加一定电压时，它发出的是红外线而不是可见光。如图 10-65 所示的就是两种红外发光二极管的实物图。



图 10-65　两种常见的红外发光二极管实物图

红外发光二极管一般有黑色、深蓝、透明 3 种颜色。判断红外发光二极管好坏的办法与判断普通二极管一样：用万用表电阻挡量一下红外发光二极管的正、反向电阻即可。红外发光二极管的驱动电路很简单，我们不妨来看看一个成品遥控器的原理，如图 10-66 所示，可以看出，红外发光二极管的驱动电路主要就是一个 NPN 型的晶体管，再辅以简单的外围即可。

3. 信号调制

早期的红外遥控中，发射的红外信号是不经过调制的，比如一种现在仍在使用的叫做 IRT1250 的编码，就是用如图 10-67 所示的编码方式。这种编码用前后两个脉冲之间的时间长度来编码“0”和“1”，而在 10μs 的信号发射期间，红外发光二极管是一直处于导通状态的。后来使用发现，采用这种方法编码的遥控器距离近，而且容易受到干扰，因为红外发光二极管不可以长时间大电流工作，否则使用寿命不长。所以一般在所加电压和导通时间之间需要找合适的平衡点，因此无载波调制的编码一般都将红外发光二极管的导通时间控制得比较小，所以使用起来效果就不好，因此后来就采用了调制和解调的技术。我们先来看看 UPD6121 编码的信号，如图 10-68 所示，是按照 UPD6121 编码发射出去的一帧数据的波形的前面部分，最左边部分叫做引导码，后面的 0 和 1 编码在红外发光二极管工作期间并不是一直导通的，而是以 38kHz 的频率间歇的导通和截至，这就使得加在红外发光二极管上的瞬时电压可以增大，这样遥控距离就提高了很多，而且编码中 0 和 1 的高电平时间也可以变大而不用担心红外发光二极管会连续工作。其实还有非常重要的一点，就是在采用了调制和解调之后，遥控器的抗干扰能力也有了大幅度提高，当然这是基于信号接收电路更复杂的前提之下的，这个在后面信号接收电路里具体说明。

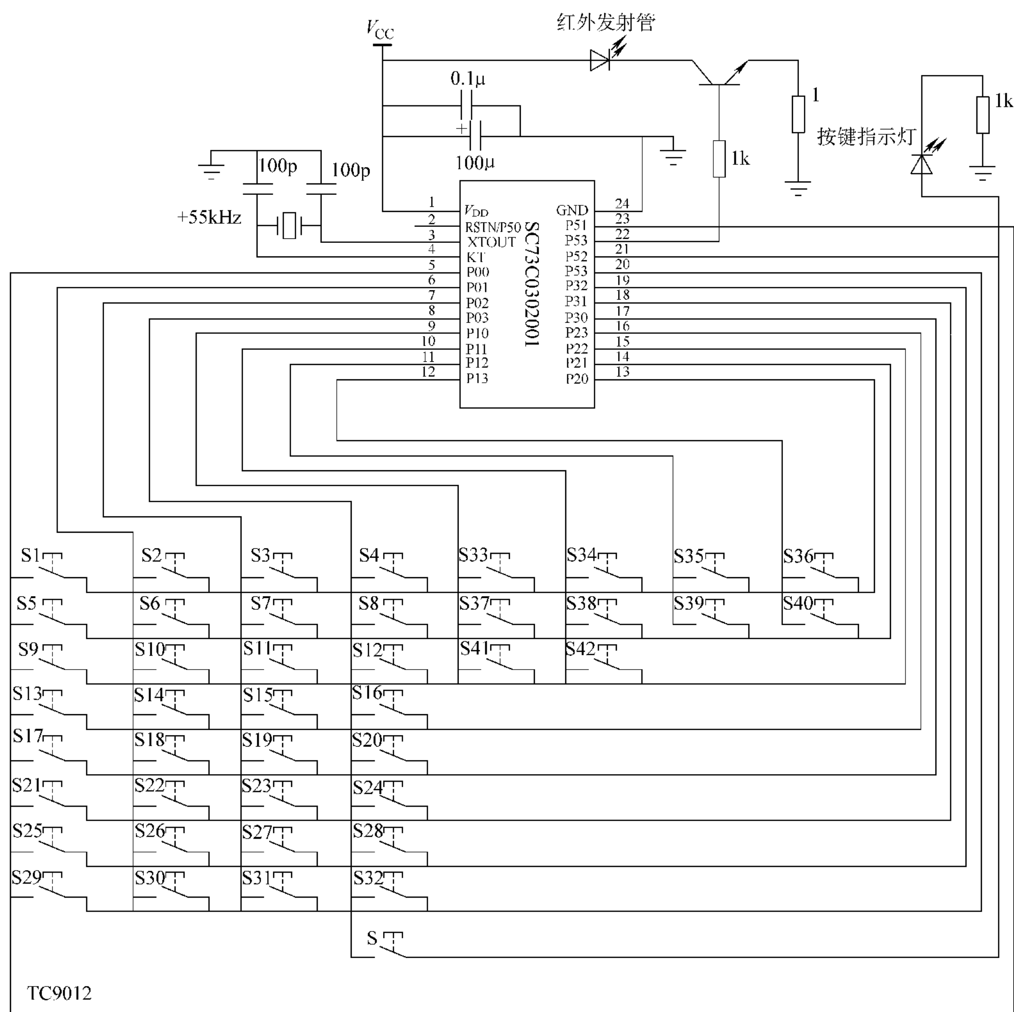


图 10-66　某成品遥控器电路原理

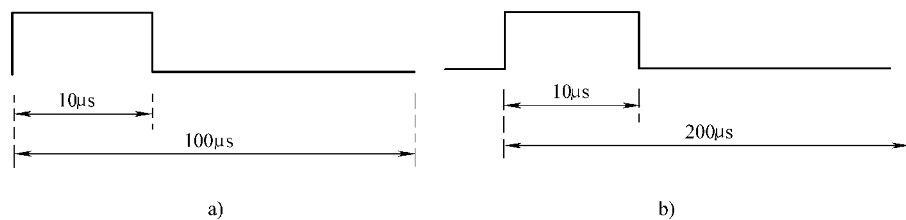


图 10-67　IRT1250 的编码

a）“0”的编码　b）“1”的编码

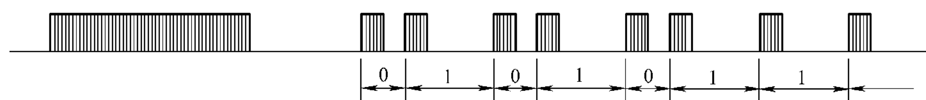


图 10-68　UPD6121 编码发射某数据时前面部分信号示意图

4. 红外信号接收电路

用来接收红外信号的器件也是一种二极管，同可见光光敏二极管相比，其特点就是对红外信号敏感，如图 10-69a 所示的就是 3 种常见的红外接收二极管的外形，它们有些什么特性呢？最重要的一点，就是当它们处在反向偏置状态时，反向电流随着受到的红外信号强度的增强而上升，所以我们可以得到图 10-69b 所示的一个简单的信号接收电路。

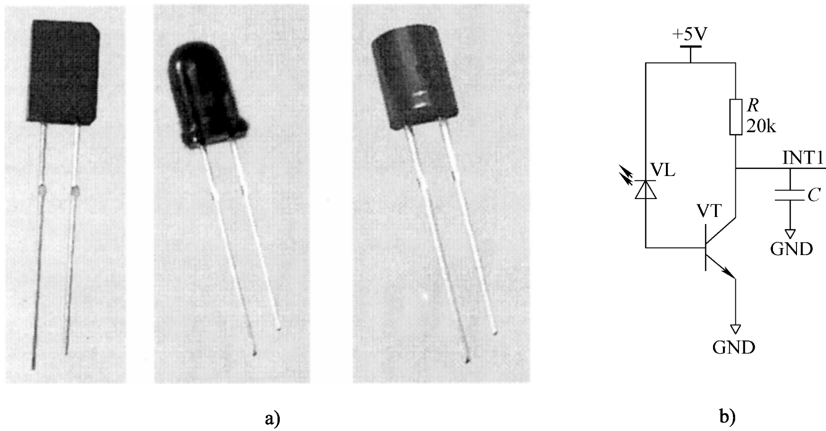


图 10-69　3 种常见的红外接收二极管实物图和简易接收电路

在图 10-69b 的接收电路里，红外接收二极管反向偏置直接串在晶体管基极和电源正端之间，这样在晶体管的集电极就可以产生一个反相变化的信号，由于红外发光二极管的发射功率一般都较小（100mW 左右），而且接收管的灵敏度也很有限，所以如果要在晶体管集电极产生满足逻辑电平条件的信号，那么这个接收电路的实际工作距离只有 10～20cm（信号由普通红外遥控器产生），这在实际使用中，是无法忍受的。为了增加距离就要增加高增益放大电路，另外因为增益的增加使得接收电路更容易受到干扰，所以也必须有相应的防干扰电路，不过这样的电路已经很成熟，例如以前的电视机的红外遥控接收电路里，有个很常用的红外接收专用集成放大电路 CX20106A，如图 10-70 所示。CX20106A 在一个电路里集成了前置放大，限幅放大，带通滤波，波形整形等电路，实际使用的时候可以调整放大倍数，带通滤波的中心频率等参数。不过因为抗干扰的需要，整个电路需要封装在一个金属屏蔽盒

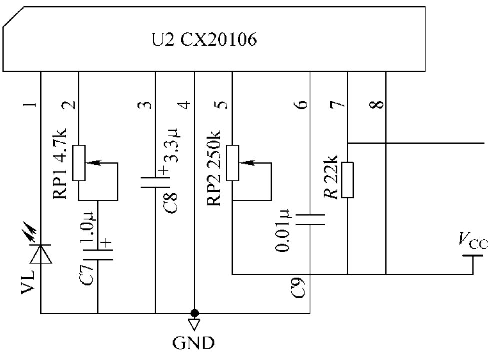




图 10-70　CX20106A 的原理和实物图

里，因而结构比较复杂，体积也不小，所以现在使用最多的是一体化的接收头，也就是把上面所有的电路都集成在一个封装里，具体见下一部分。

5. 集成红外信号接收头

最近几年不论是业余制作还是正式产品，大多都采用一体化红外接收头。一体化红外接收头的封装大致有两种：一种采用金属屏蔽；一种是塑料封装。均只有 3 只引脚，即电源正（V_{DD}）、电源负（GND）和数据输出（VO 或 OUT），不过每种封装又分大小，小的如图 10-71a 所示，大的如图 10-71b 所示。红外接收头的引脚排列因型号不同而不尽相同，可参考厂家的使用说明。一体化红外接收头的优点是不需要复杂的调试和外壳屏蔽，使用起来如同一只晶体管，非常方便，但在使用时注意成品红外接收头的载波频率。红外遥控最常用的载波频率为 38kHz，这是由于发射端使用的 455kHz 晶振决定的，不过也有一些遥控系统采用 36 kHz、40 kHz、56 kHz 等。

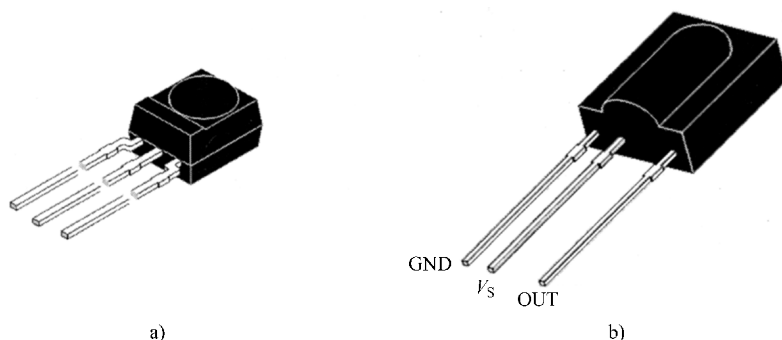


图 10-71　两种常见的红外信号接收头实物图

一体化红外接收头由于不可以调整，所以能够接收的频率是固定的，增益的调整也不再通过外部电路实现，而是在内部使用了增益自动控制（AGC）电路。TSOP173x 系列一体化红外接收头的原理框图如图 10-72 所示。

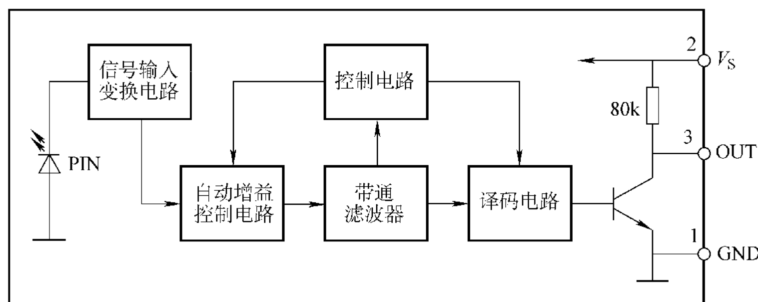


图 10-72　TSOP173x 系列一体化红外接收头的原理框图

10.9.2　红外遥控的编码和软件解码方法

前面我们已经了解了红外遥控的硬件基础，可以看出，发射电路其实只能够给出两种状态，就是有信号和无信号。为了能够表示更多的信息，就要对发出的信号进行编码，就是要把遥控指令调制成一系列的脉冲串信号。

1. 红外遥控的编码

红外遥控发射器是一种脉冲编码调制器，它在发射遥控指令时，把二进制数调制成一系列的脉冲串信号后再发射出去，常用的调制方法有脉冲宽度调制（PWM）和脉冲相位调制（PPM）两种，前面提到的 IRT1250 的编码方式就是一个很典型的脉冲宽度调制的例子，下面分别以 UPD6121（PWM）和 SAA3010（PPM）的编码来介绍这两种编码的方法。

从图 10-73 我们可以看到，UPD6121 编码方式里，把低电平宽度为 1 个周期的信号定义为逻辑 0，低电平持续 3 个周期的定义为逻辑 1，这样通过对低电平宽度的区分我们就可以识别出 0 和 1，然后就可以把遥控信号组成一个串行序列了。一帧完整的 UPD6121 的发射码由引导码、用户编码和键数据码三部分组成。引导码由 1 个 9ms 高电平脉冲及 4. 5ms 的低电平脉冲组成；8 位用户编码，被连续发送两次；8 位的键数据码也被连续发送两次，第一次发送的是键数据码的原码，第二次发送的是键数据码的反码，这样就可以在接收侧实现对数据的校验，如图 10-74 所示。

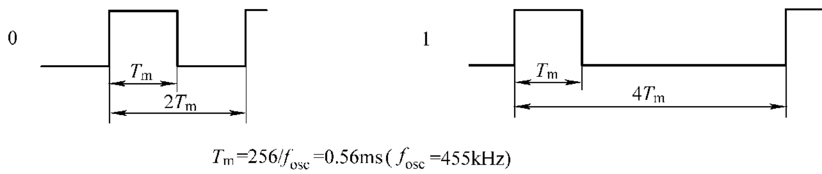


图 10-73　UPD6121 编码的逻辑 0 和 1 的定义

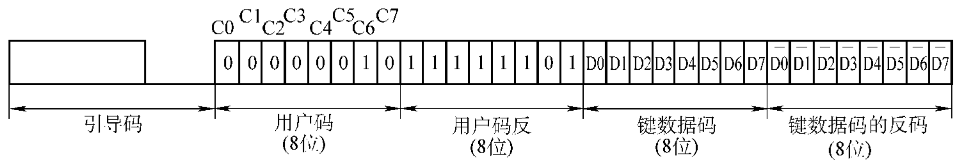


图 10-74　一帧完整的 UPD6121 的发射码构成

SAA3010（RC-5）格式的红外遥控信号是一种相位调制式脉冲编码，从图 10-75 可以看出逻辑 0 对应了从高电平到低电平的一次变化，而逻辑 1 的定义刚好与之相反。SAA3010 的控制序列有以下一些信息构成，2bit 启动位，1bit 反转位，5 bit 系统编码，6 bit 按键数据编码。其发射码的构成及实际发射的脉冲包络波形如图 10-76 所示。

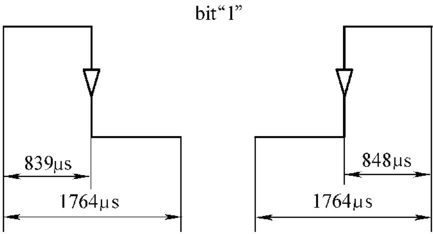


图 10-75　SAA3010 编码的逻辑 0 和 1 的定义

前面已经说过，红外遥控的编码多种多样，不过用得最多的还是脉冲宽度调制方式，就是说都类似于 UPD6121，只是有的 0 和 1 定义时高、低电平的宽度不同，也有的是没有引导码，再或者就是没有反码，总之整个规则是一样的。所以在实际使用中。可以参考具体的编码资料及针对遥控器的解码，只要掌握这两种类型的编码方式的解码，其他的也就基本类似了。

2. 红外遥控的软件解码方法

到目前为止，我们已经清楚，从遥控器过来的信号，经过接收电路处理以后，将在输出

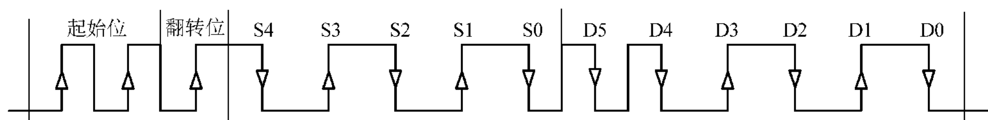


图 10-76　一帧完整的 SAA3010 的发射码构成

端给出一个脉冲序列，而解码就是从这个脉冲序列中把信息恢复出来，不过注意很重要的一点：收到的信号是发射端信号的反相电平。下面就分别以 UPD6121 和 SAA3010 为例来看看 PWM 和 PPM 遥控器编码的解码方法。

UPD6121 的编码在一体化接收头输出的信号前面部分如图 10-77 所示，9ms 低电平 +4.5ms高电平的引导码，接着就是用户码等。如果我们现在要来描述这个信息，我们会怎么说？还不就是 9ms 低电平 +4.5ms 高电平，是的，对引导码而言，只要我们能够确定低电平和高电平的时间长度在 9ms 和 4.5ms 左右就可以确定这个信息是引导码；同样对于后面的用户码和按键码，只要能够确定整个单位码长是 $2T_m$ 或者 $4T_m$ 即可确定代表的数字，然后从这串数字中，便可以恢复出遥控器发射出来的信息。总之，时间长度代表了信息，所以对 UPD6121 的编码，我们解码的途径就是获取电平的时间长度，然后按照给定参数识别出信息来。

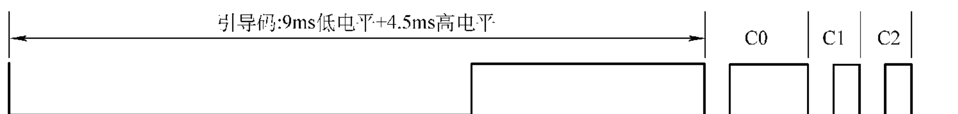


图 10-77　UPD6121 编码在一体化接收头输出的信号前面部分

目标有了，那么如何实现呢？应该很容易想到定时器吧？12MHz 晶振的时候，定时器加 1 就是 1μs，我们可以用引导码的下降沿来触发单片机的定时器开始计时处理，这样就可以获得电平的时间长度了，不过出于软硬件效率方面的取舍，仍然可以有不同的方法来实现。了解方法之前先交代一点，一般的遥控器编码在长按按键的时候，会连续发数据，可能是同样的数据，也有可能是个特定的所谓重复帧，虽然帧间的间隔大小不等，但一般在 20 ~100ms 之间，而有效的 0 和 1 的编码时间却基本小于 10ms，就是说大致上 15ms 之内没有信号收到就表示当前的数据帧已经接收完毕，所以可以有如下一些接法：

① 输出信号接单片机 INT0 端，同时该信号经过非门之后接 INT1 端，两个外部中断设置成下降沿触发，在 INT0 中断的时候开始记录低电平的宽度，在 INT1 中断的时候记录高电平的宽度，同时通过设定超时参数来检测数据接收完毕。如图 10-78a 所示。这种方法的优点是软件反应比较快，中断部分处理不占用太多时间。缺点是占用两个外部中断、两个定时器（一个用于计时处理，一个用于超时处理），还得再用一个非门，几乎快把单片机重要资源给耗光了。

② 输出信号接单片机的 INT0 和 T0 端，定时器 T0 和 T1 都初始化为定时器工作方式 1，T0 的 GATE 位置位。每次外部中断首先停止定时，记录 T0、T1 的计数值，然后将 T0、T1 的计数值清零，并重新启动定时。如图 10-78b 所示。T0 的值即为高电平脉宽，T1 - T0 的值为低电平脉宽，这样不能做超时，虽然一般而言这种方法不太会有问题，但是不做超时处理并不是很好。这种方法效率不错，但是占用了两个外部引脚，两个定时器，且不可以做超时处理。

③ 输出信号只接单片机的 INT0，引导电平的下降沿开始用 T0 计低电平的时间长度并等待电平变高，电平变高后保存 T0 并清 0，然后开始记录高电平的时间，如此不停执行，直到某个高电平过程出现超时则数据帧接收完毕，如图 10-78c 所示。这种方法占用的硬件资源最少，但软件在中断程序里逗留时间比较长，在一些时间敏感的场合不合适使用。

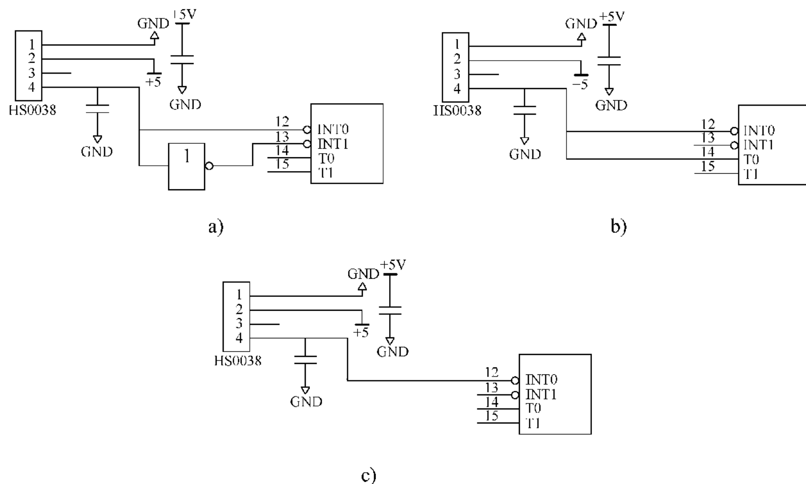


图 10-78　接收头和单片机之间的连接方法

由于实验板上采用的是上述方案③，所以下面就具体看看如何在单接 1 根中断信号线的情况下，实现解码过程。

整个过程可以分成两步，第一步是读取并判断引导码是否正确，如果不是则直接返回并初始化检测参数；第二步是连续 4 次，按照每次 8bit 读取 4 个字节的后续数据，其中，检测过程中对 0 和 1 的判断必须是在开启计时之后，以减少因为程序执行而导致的测量时间长度上的误差。具体解码流程如图 10-79 所示。

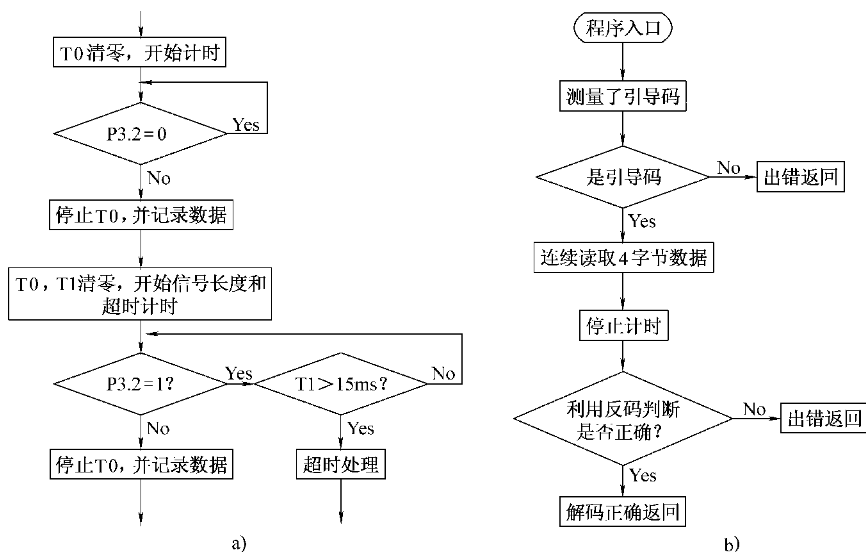


图 10-79　UPD6121 遥控器编码的解码子程序流程

a）一个位的识别流程　b）整个编码的识别流程

通过对 UPD6121 编码的解码，相信你已经掌握了基本方法了，不过 SAA3010 的解码同 UPD6121 还是不同的，这种类型的编码除了本身编码方式特别，并且在发射的数据不同的时候，也会有需要注意的不同点，首先关注一下图 10-80 中开始处和末尾处的实线和圆圈标记，这是两个特点所在，下面具体分析。

图 10-80a 给出的是一个发送端实际发射的脉冲包络波形，实际发射的波形本来是有 2 个起始位的，就是说有两个“1”，但是一般资料都说是 1.5 个起始位，为什么？我们可以看图 10-80b，因为通常接收端平常保持高电平，所以发送的第一个 bit“1”时的起始低电平部分在接收端就不能表现出来，所以真正有效的起始信号是从高电平开始的，也就是第一个 bit“1”的后半部分，因此就有了 1.5 个起始位的说法。

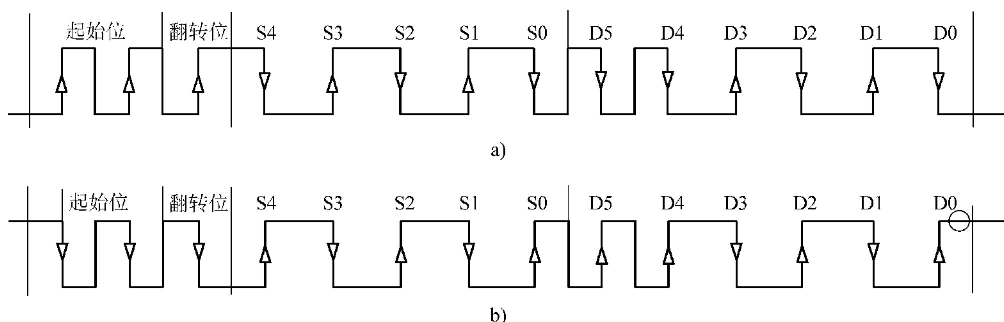


图 10-80　SAA3010 编码的发射接收波形

a）SAA3010 遥控器发射端的波形，系统码 0x0A，按键码 0x0A

b）SAA3010 编码在接收头输出的波形，系统码 0x0A，按键码 0x0A

再看波形的最右边部分，由于按键编码是 0x0A，所以最低位就是 0，这样最后的一个 bit 在接收端表现出来的高电平也将检测不到，这在解码过程中需要注意。

与 UPD6121 不同，我们还无法只从时间长度上识辨编码，因为编码的本意关注的就是跳变，所以我们可以从低电平到高电平和高电平到低点平这样的一对过程来确定编码。观察波形和图 10-75 中 SAA3010 编码的位定义，现在我们如果把 850μs 左右的时长定为一个有效电平的话，那么 0 的定义不就是“10”，而 1 的定义就变成了“01”？从波形可以看出，不管连续两个位置是任何数据，中间高低电平的长度只有两种可能，就是在 850μs 和 1700μs 左右，既然这样，我们就利用两个时间的中间值来判断一个电平里包含的是一个还是两个电平，比如我们可以从图 10-80b 中可以获得这样一个序列：

010 10 0110011001 010110011001

由于前面的 1.5bit 是起始位，一直保持不变（注意，有极少数 SAA3010 的编码起始位定义不一样，以具体资料为准），所以前面的 010 就是固定的，不需要理会。因为看的是接收端波形，所以我们把图 10-75 中 1 和 0 的位定义反过来看，这样，翻转位就是 1（10），而系统码是 01010（0110011001），按键码是 001010（010110011001），到此为止，我们已经把数据从脉冲串里恢复出来了。具体的程序流程如图 10-81 所示。

10.9.3　遥控器软件解码的程序实现

1. UPD6121 遥控器软件解码程序

```
#define uPD6121_GLOBALS
```

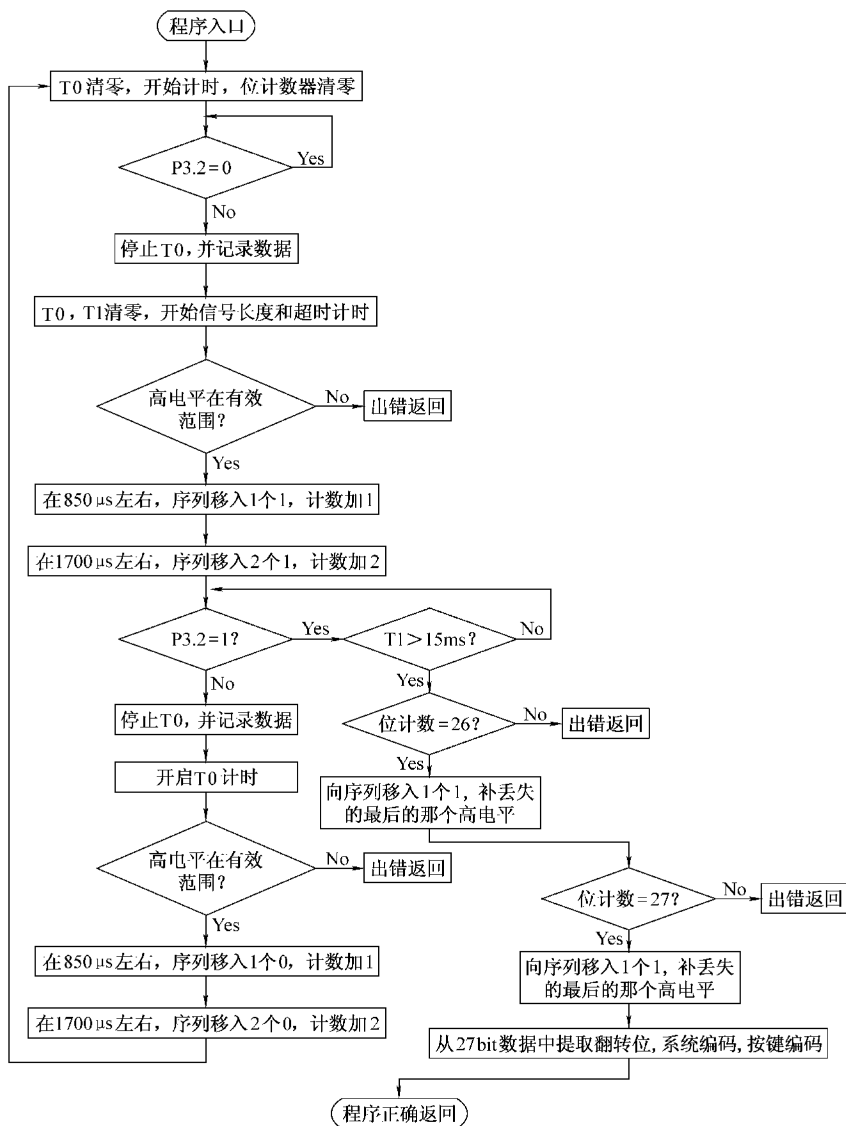


图 10-81　SAA3010 遥控器编码的解码子程序流程

```
#include <reg52. h>
#include "uPD6121. h"
#include "main. h"

//========================================
// 从 ir_receive 接收连续的 8 个高低电平，并根据参数返回解码出来的一个字节数据
// 适合对脉宽调制类型遥控的解码，高低电平是针对编码而言的，对解码过程则相反
//----------------------------------------
#pragma AREGS
unsigned char uPD6121_read_code_8 (void)
```

```
{
    unsigned char temp =0;
    unsigned char i;

    for (i =0; i <8; i + + )
    {
        temp = temp >>1;
        while (ir_receive = =0);                             // 等待高电平测试结束
        TR1 =0;                                              // 高电平测试结束，停止计时
        high_level_time = TH1 * 256 + TL1;                   // 保存高电平的数据

        TH1 =0; TL1 =0; TR1 =1;                              // 启动对低电平的测试
        while (ir_receive = =1)
          {
            if (TH1 >0x4e) { decode_error =1; return 0;}  // 超时出错，返回
          }

        TR1 =0;                                              // 低电平测试结束
        low_level_time = TH1 * 256 + TL1;                    // 保存低电平的数据

        TH1 =0; TL1 =0; TR1 =1;                              // 如果电平长度不在合理的范
                                                               围内，则认为出错
    if ((high_level_time <400) | | (high_level_time >700) | | (low_level_time <400)
| | (low_level_time >1900))
          {
            decode_error =1;
            return 0;
          }
        if ((low_level_time >400) && (low_level_time <700)) temp = temp&0x7f;
        if ((low_level_time >1400) && (low_level_time <1900)) temp = temp | 0x80;
    }
    return temp;
}
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// 解码出错返回 1，对则返回 0
#pragma AREGS
unsigned char uPD6121_decode (void)
```

```
{
//-----------------------------------------
    TR1 =1;                                    //开始测量引导码（或重复码）
                                                 的高电平宽度
    while（ir_receive = =0）;                  //等待电平变高，不需要超时
                                                 监测
    TR1 =0;                                    //高电平（对发射电路而言）
                                                 测试结束
    high_level_time = TH1 * 256 + TL1;         // 保存高电平的数据
//-----------------------------------------
    TH1 =0; TL1 =0; TR1 =1;                    // 启动对低电平的测试
    while（ir_receive = =1）                   //在对高电平进行查询时，计
                                                 超时，大于20ms出错
    {
        if（TH1 >0x4e）return 1;               //测试超时后直接初始化相关
                                                 变量，开始下次测试
    }
    TR1 = 0;                                   //低电平（对发射电路而言）
                                                 测试结束
    low_level_time = TH1 * 256 + TL1;          //保存低电平的数据

    TH1 =0; TL1 =0; TR1 =1;                    //为增加计时的准确性，数据
                                                 的处理都是在计时过程里
// 判断引导码（或重复码）是否正确，如果不正确，则设置出错标志位，并退出中断程序
    if（(high_level_time < 8500）||（high_level_time > 9500）||（low_level_time <
1000）||（low_level_time >5000））
    {
        return 1;                              //因是引导码出错，所以直接
                                                 初始化后重新开始测试
    }
//-----------------------------------------
// 对是引导码还是重复码进行判断，如果是重复码，就跳过后面数据的读取
    if（(low_level_time >1000）&&（low_level_time <3500））repeat_code_detected =1;
    if（(low_level_time >4000）&&（low_level_time <5000））load_code_detected =1;

    if（repeat_code_detected = =1）return 1;   // 直接结束,
```

```
    temp1 = uPD6121_read_code_8 ();                    // 读后面的系统、按键等数据,
    temp2 = uPD6121_read_code_8 ();
    temp3 = uPD6121_read_code_8 ();
    temp4 = uPD6121_read_code_8 ();
    TR1 = 0;

    if (decode_error == 1) return 1;                   //无论是哪部分解码出错,都
                                                         是重新开始
    if ((temp1 != ~temp2) || (temp3 != ~temp4)) { return 1; }

    sys_code = temp1;
    key_code = temp3;
    data_available = 1;
    return 0;
}
```

2. SAA3010 遥控器软件解码程序

```
    #define SAA3010_GLOBALS
    #include <reg51.h>
    #include "SAA3010.h"
    #include "main.h"

//========================================
// 该函数的作用是每调用一次就在 temp1 ~4 组成的 32bit 长度的最低位上移入
// 一个 0 或者 1,数据由 bitdata 确定
//----------------------------------------
void SAA3010_cycle_data (unsigned char bitdata)
{
    temp4 = temp4 << 1;
    if ((temp3&0x80) == 1) temp4 = temp4 | 0x01;
    else temp4 = temp4&0xfe;

    temp3 = temp3 << 1;
    if ((temp2&0x80) == 1) temp3 = temp3 | 0x01;
    else temp3 = temp3&0xfe;

    temp2 = temp2 << 1;
    if ((temp1&0x80) == 1) temp2 = temp2 | 0x01;
```

```
    else temp2 = temp2&0xfe;

    temp1 = temp1 < <1;
    if (bitdata = =1) temp1 = temp1 | 0x01;
    else temp1 = temp1&0xfe;
}

// 解码出错返回 1，对则返回 0
unsigned char SAA3010_decode (void)
{
//------------------------------------------
    unsigned char count =0;

    TR1 =1;                                        // 启动计时
    while (1)
    {
    while (ir_receive = =0);                       //等待电平变高，不需要超时
                                                     监测
    TR1 =0;                                        //高电平（对发射电路而言）
                                                     测试结束
    high_level_time = TH1 * 256 + TL1;             // 记录高电平的数据
//------------------------------------------
    TH1 =0; TL1 =0; TR1 =1;                        // 启动对低电平的测试
//------------------------------------------
// 处理低电平
        if ( (high_level_time <750) | | (high_level_time
 >1800)) return 1;                                 // 不是合格的电平

    if ((high_level_time >750) && (high_level_time
 <1000)) {SAA3010_cycle_data (0); count + =1;}     // 移入一个 0
        if ((high_level_time >1500) && (high_level_time
 <1800)) {SAA3010_cycle_data (0); SAA3010_cycle_data (0); count + =2;}
                                                   // 移入两个 0

    while (ir_receive = =1)                        // 等待电平变低
    {
        if (TH1 >0x08) break;                      //高电平超时，正常时是测试
                                                     结束，异常时是出错
    }
```

```
    TR1 = 0;                                              //低电平（对发射电路而言）
                                                            测试结束
    if (TH1 > 0x08) {break;}
    low_level_time = TH1 * 256 + TL1;                     // 保存低电平的数据

    TH1 = 0; TL1 = 0; TR1 = 1;                            //为增加计时的准确性，数据
                                                            处理都是在计时过程里
//-------------------------------------------------------------------------
// 处理高电平
      if ((low_level_time < 750) || (low_level_time > 1800)) return 1;
                                                          // 不是合格的电平
      if ((low_level_time > 750) && (low_level_time < 1000)) {SAA3010_cycle_data
(1); count += 1;}                                         // 移入一个0
      if ((low_level_time > 1500) && (low_level_time < 1800)) {SAA3010_cycle_data
(1); SAA3010_cycle_data (1); count += 2;}                 // 移入两个0
    }

    if (count == 26) {SAA3010_cycle_data (1); count++;}
    if (count != 27) return 1;
    led = 0;

// 提取按键信息
    key_code = 0;
    if ((temp1 >> 1) & 0x01) key_code = key_code | 0x01;
    else key_code = key_code & 0xfe;
    if ((temp1 >> 3) & 0x01) key_code = key_code | 0x02;
    else key_code = key_code & 0xfd;
    if ((temp1 >> 5) & 0x01) key_code = key_code | 0x04;
    else key_code = key_code & 0xfb;
    if ((temp1 >> 7) & 0x01) key_code = key_code | 0x08;
    else key_code = key_code & 0xf7;
    if ((temp2 >> 1) & 0x01) key_code = key_code | 0x10;
    else key_code = key_code & 0xef;
    if ((temp2 >> 3) & 0x01) key_code = key_code | 0x20;
    else key_code = key_code & 0xdf;

// 提取系统信息
    sys_code = 0;
```

```
    if ((temp2 >>5) &0x01) sys_code = sys_code | 0x01;
    else sys_code = sys_code&0xfe;
    if ((temp2 >>7) &0x01) sys_code = sys_code | 0x02;
    else sys_code = sys_code&0xfd;

    if ((temp3 >>1) &0x01) sys_code = sys_code | 0x04;
    else sys_code = sys_code&0xfb;
    if ((temp3 >>3) &0x01) sys_code = sys_code | 0x08;
    else sys_code = sys_code&0xf7;
    if ((temp3 >>5) &0x01) sys_code = sys_code | 0x10;
    else sys_code = sys_code&0xef;
    if ((temp3 >>7) &0x01) sys_code = sys_code | 0x20;
    else sys_code = sys_code&0xdf;

    data_available = 1;
    return 0;
}
```

10.10　模/数转换器应用实例

在工业控制和智能化仪表中，通常由微型计算机进行实时控制及实时数据处理。计算机所加工的信息总是数字量，而被控制或被测量的有关参量往往是连续变化的模拟量，如温度、速度、压力等，与此对应的电信号是模拟信号。模拟量的存储和处理比较困难，不适合作为远距离传输且易受干扰。在一般的工业应用系统中传感器把非电量的信号变成与之对应的模拟信号，然后经模拟（Analog）到数字（Digital）转换电路将模拟信号转成对应的数字信号送计算机处理。这就是一个完整的信号链，模拟到数字的转换过程就是我们经常接触到的 ADC（Analog to Digital Convert）电路。

10.10.1　模/数转换器简介

1. 模/数转换原理

ADC 的转换原理根据 ADC 的电路形式有所不同。ADC 电路通常由两部分组成，它们是：采样、保持电路和量化、编码电路。其中量化、编码电路是最核心的部件，任何 ADC 转换电路都必须包含这种电路。ADC 电路的形式很多，通常可以并为两类：

间接法：它是将采样-保持的模拟信号先转换成与模拟量成正比的时间或频率，然后再把它转换为数字量。通常采用时钟脉冲计数器，又被称为计数器式。它的工作特点是：工作速度低，转换精度高，抗干扰能力强。

直接法：通过基准电压与采样-保持信号进行比较，从而转换为数字量。它的工作特点是：工作速度高，转换精度容易保证。

模/数转换的过程有四个阶段，即采样、保持、量化和编码。

采样是将连续时间信号变成离散时间信号的过程。经过采样，时间连续、数值连续的模拟信号就变成了时间离散、数值连续的信号，称为采样信号。采样电路相当于一个模拟开关，模拟开关周期性地工作。理论上，每个周期内，模拟开关的闭合时间趋近于0。在模拟开关闭合的时刻（采样时刻），我们就“采”到模拟信号的一个“样本”。

量化是将连续数值信号变成离散数值信号的过程。理论上，经过量化，我们就可以将时间离散、数值连续的采样信号变成时间离散、数值离散的数字信号。

我们知道，在电路中，数字量通常用二进制代码表示。因此，量化电路的后面有一个编码电路，将数字信号的数值转换成二进制代码。

然而，量化和编码总是需要一定时间才能完成，所以，量化电路的前面还要有一个保持电路。保持是将时间离散、数值连续的信号变成时间连续、数值离散信号的过程。在量化和编码期间，保持电路相当于一个恒压源，它将采样时刻的信号电压“保持”在量化器的输入端。虽然逻辑上保持器是一个独立的单元，但是，工程上保持器总是与采样器做在一起。两者合称采样保持器。

2. 模/数转换分类

下面简要介绍几种常用的A/D转换器的基本原理及特点：积分型、逐次比较型、并行比较型/串并行比较型、Σ-Δ调制型、电容阵列逐次比较型及压频变换型。

1）积分型：积分型A/D转换器工作原理是将输入电压转换成时间（脉冲宽度信号）或频率（脉冲频率），然后由定时器/计数器获得数字值。其优点是用简单电路就能获得高分辨率，但缺点是由于转换精度依赖于积分时间，因此转换速率极低。初期的单片A/D转换器大多采用积分型，现在逐次比较型已逐步成为主流。

2）逐次比较型：逐次比较型A/D转换器由一个比较器和D/A转换器通过逐次比较逻辑构成，从MSB开始，顺序地将每一位输入电压与内置D/A转换器输出进行比较，经n次比较而输出数字值。其电路规模属于中等。其优点是速度较高、功耗低，在低分辨率（<12位）时价格便宜，但高精度（>12位）时价格很高。

3）并行比较型/串并行比较型：并行比较型A/D转换器采用多个比较器，仅作一次比较而实行转换，又称快速（FLash）型。由于转换速率极高，n位的转换需要$2n-1$个比较器，因此电路规模也极大，价格也高，只适用于视频A/D转换器等速度特别高的领域。

串并行比较型A/D转换器结构上介于并行比较型和逐次比较型之间，最典型的是由两个$n/2$位的并行型A/D转换器配合D/A转换器组成，用两次比较实行转换，所以称为半快速（Half flash）型。还有分成三步或多步实现A/D转换的叫做分级（Multistep/Subrangling）型A/D转换器，而从转换时序角度又可称为流水线（Pipelined）型A/D转换器，现代的分级型A/D转换器中还加入了对多次转换结果作数字运算而修正特性等功能。这类A/D转换器速度比逐次比较型高，电路规模比并行比较型小。

4）Σ-Δ（Sigma-delta）调制型：Σ-Δ型A/D转换器由积分器、比较器、1位D/A转换器和数字滤波器等组成。原理上近似于积分型，将输入电压转换成时间（脉冲宽度）信号，用数字滤波器处理后得到数字值。电路的数字部分基本上容易单片化，因此容易做到高分辨率。主要用于音频和测量。

5）电容阵列逐次比较型：电容阵列逐次比较型A/D转换器在内置D/A转换器中采用电容阵列方式，也可称为电荷再分配型。一般的电阻阵列D/A转换器中多数电阻的值必须

一致，在单芯片上生成高精度的电阻并不容易。如果用电容阵列取代电阻阵列，可以用低廉成本制成高精度单片 A/D 转换器。最近的逐次比较型 A/D 转换器大多为电容阵列式的。

6）压频变换型：压频变换型（Voltage-Frequency Converter）是通过间接转换方式实现 A/D 转换的。其原理是首先将输入的模拟信号转换成频率，然后用计数器将频率转换成数字量。从理论上讲这种 A/D 转换器的分辨率几乎可以无限增加，只要采样的时间能够满足输出频率分辨率要求的累积脉冲个数的宽度。其优点是分辨率高、功耗低、价格低，但是需要外部计数电路共同完成 A/D 转换。

10.10.2　A/D 转换器的主要技术指标

1）分辨率（Resolution）：指数字量变化一个最小量时模拟信号的变化量，定义为满刻度与 $2n$ 的比值。分辨率又称精度，通常以数字信号的位数来表示。

2）转换速率（Conversion Rate）：是指完成一次从模拟转换到数字的 A/D 转换所需的时间的倒数。积分型 A/D 转换器的转换时间是毫秒级属低速 A/D 转换器，逐次比较型 A/D 转换器是微秒级属中速 A/D 转换器，全并行/串并行型 A/D 转换器可达到纳秒级。采样时间则是另外一个概念，是指两次转换的间隔。为了保证转换的正确完成，采样速率（Sample Rate）必须小于或等于转换速率。因此有人习惯上将转换速率在数值上等同于采样速率也是可以接受的。常用单位是 ksps 和 Msps，表示每秒采样千/百万次（kilo / Million Samples per Second）。

3）量化误差（Quantizing Error）：由于 A/D 转换器的有限分辨率而引起的误差，即有限分辨率 A/D 转换器的阶梯状转移特性曲线与无限分辨率 A/D 转换器（理想 A/D 转换器）的转移特性曲线（直线）之间的最大偏差。通常是 1 个或半个最小数字量的模拟变化量，表示为 1LSB、1/2LSB。

4）偏移误差（Offset Error）：输入信号为零时输出信号不为零的值，可外接电位器调至最小。

5）满刻度误差（Full Scale Error）：满度输出时对应的输入信号与理想输入信号值之差。

6）线性渡（Linearity）：实际转换器的转移函数与理想直线的最大偏移，不包括以上三种误差。

其他指标还有：绝对精度（Absolute Accuracy），相对精度（Relative Accuracy），微分非线性，单调性和无错码，总谐波失真（Total Harmonic Distotortion 缩写 THD）和积分非线性。

10.10.3　串行 A/D 转换器 ADC0832 简介

ADC0832 是美国国家半导体公司生产的一种 8 位分辨率、双通道 A/D 转换芯片。由于它体积小，兼容性强，性价比高而深受单片机爱好者及企业欢迎，其目前已经有很高的普及率。ADC083X 是市面上常见的串行模/数转换器件系列。ADC0831、ADC0832、ADC0834、ADC0838 是具有多路转换开关的 8 位串行 I/O 模/数转换器，转换速度较高（转换时间 32μs），单电源供电，功耗低（15mW），适用于各种便携式智能仪表。本节以 ADC0832 为例，介绍其使用方法。

ADC0832 是 8 引脚双列直插式双通道 A/D 转换器，能分别对两路模拟信号实现模/数转换，可以用在单端输入方式和差分方式下工作。ADC0832 采用串行通信方式，通过 DI 数据输入端进行通道选择、数据采集及数据传送。8 位的分辨率（最高分辨可达 256 级），可以适应一般的模拟量转换要求。其内部电源输入与参考电压的复用，使得芯片的模拟电压输入在 0 ~5V 之间。具有双数据输出可作为数据校验，以减少数据误差，转换速度快且稳定性能强。独立的芯片使能输入，使多器件挂接和处理器控制变的更加方便，其芯片引脚排列如图 10-82 所示。

ADC0832 具有以下特点：

1）8 位分辨率；

2）双通道 A/D 转换；

3）输入输出电平与 TTL/CMOS 相兼容；

4）5V 电源供电时输入电压在 0 ~5V 之间；

5）工作频率为 250kHz，转换时间为 32μs；

6）一般功耗仅为 15mW；

7）8P、14P—DIP（双列直插）、PICC 多种封装；

8）商用级芯片温度范围为 0° ~ +70°C，工业级芯片温度范围为 −40 ~ +85°C。

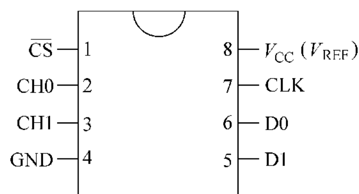


图 10-82　ADC0832 的引脚排列

芯片接口说明：

1）$\overline{CS}$：片选使能，低电平芯片使能。

2）CH0：模拟输入通道 0，或作为 IN +/ − 使用。

3）CH1：模拟输入通道 1，或作为 IN +/ − 使用。

4）GND：芯片参考零电位（地）。

5）DI：数据信号输入，选择通道控制。

6）DO：数据信号输出，转换数据输出。

7）CLK：芯片时钟输入。

8）V_{cc}(V_{REF})：电源输入及参考电压输入（复用）。

ADC0832 的时序如图 10-83 所示。

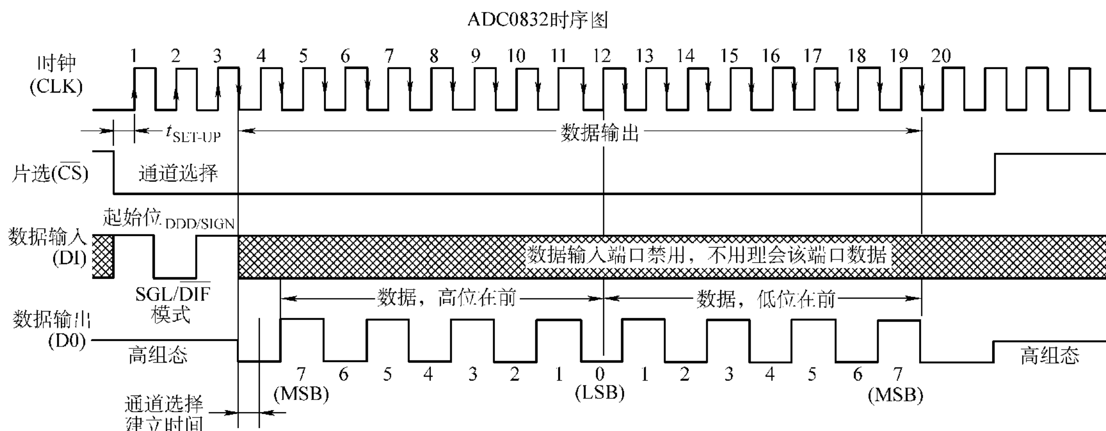


图 10-83　ADC0832 的时序

ADC0832 的控制原理

正常情况下 ADC0832 与单片机的接口应为 4 条数据线，分别是 CS、CLK、DO、DI。但由于 DO 端与 DI 端在通信时并未同时使用并与单片机的接口是双向的，所以在 I/O 口资源紧张时可以将 DO 和 DI 并联在一根数据线上使用。当 ADC0832 未工作时其 CS 输入端应为高电平，此时芯片禁用，CLK 和 DO/DI 的电平可任意。当要进行 A/D 转换时，须先将 CS 使能端置于低电平并且保持低电平直到转换完全结束。此时芯片开始转换工作，同时由处理器向芯片时钟（CLK）输入端输入时钟脉冲，DO/DI 端则使用 DI 端输入通道功能选择的数据信号。在第 1 个时钟脉冲的下沉之前 DI 端必须是高电平，表示启始信号。在第 2、3 个脉冲下沉之前 DI 端应输入两位数据用于选择通道功能。

通道地址设置见表 10-15。

如表 10-15 所示，当此两位数据为“1”、“0”时，只对 CH0 进行单通道转换。当两位数据为“1”、“1”时，只对 CH1 进行单通道转换。当两位数据为“0”、“0”时，将 CH0 作为正输入端 IN +，CH1 作为负输入端 IN－进行输入。当两位数据为“0”、“1”时，将 CH0 作为负输入端 IN－，CH1 作为正输入端 IN + 进行输入。到第 3 个脉冲的下降之后 DI 端的输入电平就失去输入作用，此后 DO/DI 端则开始利用数据输出 DO 进行转换数据的读取。从第 4 个脉冲下降沿开始由 DO 端输出转换数据最高位 Data7，随后每一个脉冲的下降沿 DO 端输出下一位数据。直到第 11 个脉冲时发出最低位数据 Data0，一个字节的数据输出完成。也正是从此位开始输出下一个相反字节的数据，即从第 11 个字节的下降沿输出 Data0。随后输出 8 位数据，到第 19 个脉冲时数据输出完成，也标志着一次 A/D 转换的结束。最后将 CS 置高电平禁用芯片，直接将转换后的数据进行处理就可以了。

表 10-15　通道地址设置

通道地址		通道		工作方式说明
SGL/DIF	ODD/SIGN	0	1	
0	0	+	－	差分方式
0	1	－	+	
1	0	+		单端输入方式
1	1		+	

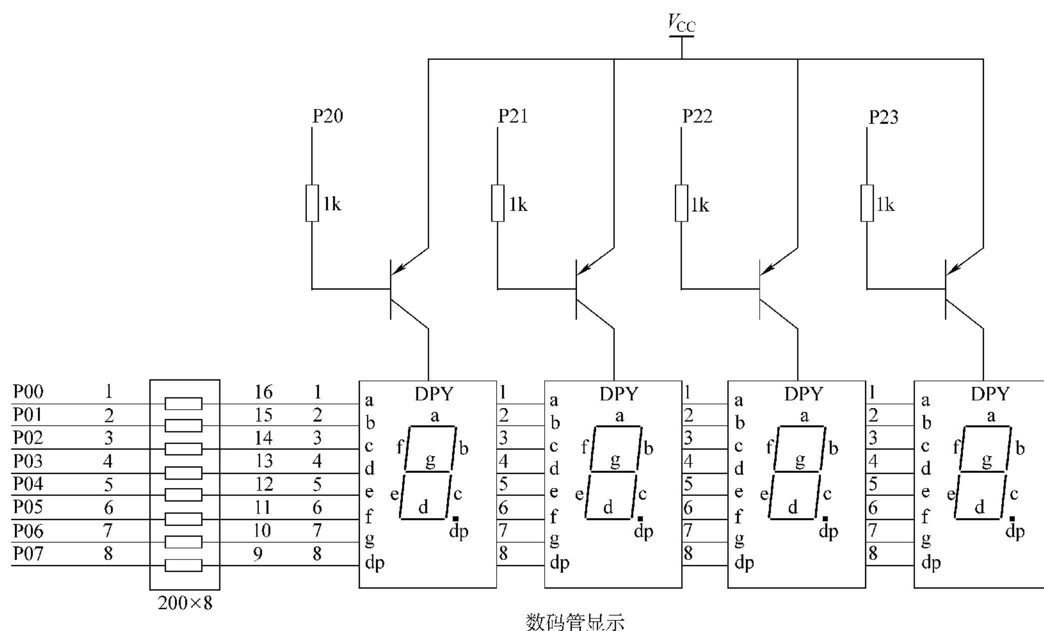
作为单通道模拟信号输入时 ADC0832 的输入电压是 0～5V 且 8 位分辨率时的电压精度为 19.53mV，即（5/256）V。如果作为由 IN + 与 IN－输入时，可以将电压值设定在某一个较大范围之内，从而提高转换的宽度。但值得注意的是，在进行 IN + 与 IN－的输入时，如果 IN－的电压大于 IN + 的电压则转换后的数据结果始终为 00H。

10.10.4　ADC0832 应用实例

通过以上的理论学习之后，对模/数转换应该有了一定的了解，接下来就根据上文的指导现场对 ADC0832 进行应用，以加深印象。

1）功能说明：将通道 0 上采样到的电压显示在 LED 数码管上，通过改变通道 0 的输入电压变化，观察输出读数。

硬件原理如图 10-84 所示。软件流程如图 10-85 所示。



数码管显示

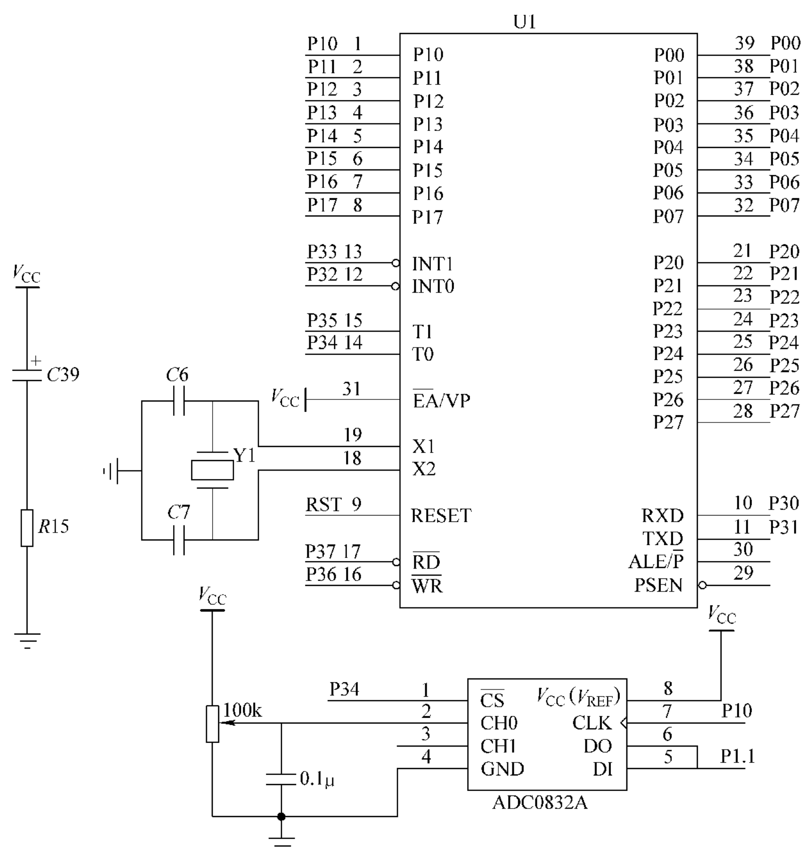


图 10-84　硬件原理

开始

端口初始化

AD初始化

读取AD值

显示

图 10-85　软件流程图

2）软件代码：

```
/***************************************************************************/
/*                         杭州电子 & 计算机工作室                          */
/*                         http://www.hificat.com                          */
/*                         ADC0832 测试程序                                 */
/*                         2006 年 4 月 20 日                               */
/*                         目标器件：AT89C51                                */
/*                         晶振：12MHz                                      */
/*                         编译环境：Keil 7.50A                             */
/***************************************************************************/
/*********************************包含头文件********************************/
#include "reg51.h"

/*********************************端口定义**********************************/
sbit CS  = P3^4;
sbit Clk  = P1^0;
sbit DIO  = P1^1;

/*********************************定义全局变量*******************************/
unsigned char adval;                    //AD 值

/*******************************共阳 LED 段码表*****************************/
unsigned char code tab [ ] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80,
0x90};

/****************************************************************************
函数功能：AD 转换子程序
入口参数：CH
出口参数：adval
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
unsigned char A_D (bit CH)
{
    unsigned char i;
    CS = 0;                              //片选
    Clk = 0;                             //时钟拉低
    Clk = 1;                             //时钟拉高
    DIO = 1;                             //启动信号
    if ( CH = = 0 )                      //通道选择
    {
        Clk = 0;
        Clk = 1;
        DIO = 1;                         //通道 0 的第一位

        Clk = 0;
        Clk = 1;
        DIO = 0;                         //通道 0 的第二位
    }
    else
    {
        Clk = 0;
        Clk = 1;
        DIO = 1;                         //通道 1 的第一位

        Clk = 0;
        Clk = 1;
        DIO = 1;                         //通道 1 的第二位
    }

    Clk = 1;
    Clk = 0;

    for ( i = 0; i < 8; i + + )          //读取八位 A/D 值
    {
        Clk = 1;
        Clk = 0;
        if ( DIO )
            adval = (adval >>1) | 0x80;
        else
```

```
            adval = (adval >>1) | 0x00;
        }
    CS = 1;                                  //释放 ADC0832
    return (adval);                          //返回采样值
}

/*****************************************
函数功能：数码管显示子程序
入口参数：
出口参数：
*****************************************/
void delay (void)
{
    int k;
    for (k=0; k<600; k++);
}

/*****************************************
函数功能：数码管显示子程序
入口参数：k
出口参数：
*****************************************/
void display (int k)
{
    P2 =0xfe;
    P0 =tab [k/1000];
    delay1 ();
    P2 =0xfd;
    P0 =tab [k%1000/100];
    delay1 ();
    P2 =0xfb;
    P0 =tab [k%100/10];
    delay1 ();
    P2 =0xf7;
    P0 =tab [k%10];
    delay1 ();
    P2 =0xff;
}
```

```
/****************************************
函数功能：主程序
入口参数：
出口参数：
****************************************/
void main (void)
{
    P2 =0xff;                           //端口初始化
    P0 =0xff;
    while (1)                           //主循环
        {
        A_D (0x00);                     //通道 0 转换
        display (adval);                //显示 AD 值
        }
}
```

10.11　DS1302 的应用

DS1302 是美国 DALLAS 公司推出的一款高性能、低功耗、带内部 RAM 的实时时钟芯片（RTC），可想而知，就是一种能够为单片机系统提供日期和时间的芯片。我们最容易想到用它来做什么呢？当然是数字钟了！不过一般的简易数字钟其实并不会采用类似的集成电路，主要还是通过对特定频率比如 60Hz 信号进行计数、译码，然后驱动数字显示器件来实现。DS1302 有它更适合使用的场合，如某些钟控设备、数据记录仪表，这些仪表往往需要采集带时标的数据，同时一般它们也会有一些需要保存起来的重要数据，DS1302 的 RAM 正是为这些数据准备的。不过在实际需要应用 RTC 的场合，你并非只有这个选择，因为无论是 DALLAS 还是其他一些集成电路生产商都提供了比 DS1302 更好、更方便使用的 RTC 芯片，但是总的来说 RTC 芯片还是具有一些共性。通过这节，我们将会把 RTC 相关的一些技术都粗略介绍一下，然后实现单片机跟 DS1302 之间的程序接口。

10.11.1　实时时钟芯片概述

1. 实时时钟芯片相关技术

实时时钟芯片（RTC）的主要功能是完成年、月、周、日、时、分、秒的计时，通过外部接口为单片机系统提供日历和时钟，所以一个最基本的实时时钟芯片通常会具有如下的一些部件：电源电路、时钟信号产生电路、实时时钟、数据存储器、通信接口电路、控制逻辑电路等，同时大部分的 RTC 还会提供一些额外的 RAM，RTC 的基本组成如图 10-86 所示。

如果直接利用单片机的定时器，是不是也可以用软件自己来写时钟、日历程序？是的，但是会有几个问题，首先为了使时钟不至于停走，就得在停电时给单片机供电，而相对 RTC 来说，单片机的功耗大很多，电池往往无法长时间工作；其次单片机计时的准确度比较差，通常很难达到需要的精度，因此目前 RTC 的使用已经十分广泛。

由于在需要 RTC 的场合，一般不允许时钟停走，所以即使在单片机系统停电的时候，RTC 也必须能正常工作，因此一般都需要电池供电，同时考虑到电池使用寿命，所以有不少 RTC 把电路设计成能够根据主电源电压的有无，自动切换 RTC 使用主电源或备用电池，即当断电的时候，后备电池能够自动给 RTC 供电，而像 DS1302 还增加了电池充电电路，用来和可充电锂电池接口。

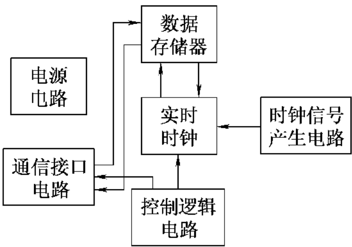


图 10-86　RTC 的基本组成

从上面的叙述里，我们看到最明显的一点是精度的差别。RTC 在使用过程中是如何控制精度的呢？一般 RTC 都是使用 32768Hz 的晶振，本身误差小，同时很多设备在生产过程中对这个频率进行过校准，主要方法就是改变两个从晶振引脚到地的电容值的大小，通过测试 RTC 输出的秒信号的频率，然后把电容改成合适的数值，使精度控制在合理的范围里，当然目前也有些时钟芯片在片内内置了电容阵列，可以自动调整。影响精度还有另外一个原因，就是温度，因此有很多产品在采用无内置温补电路的时候，会使用软件对计时进行温度补偿。当然，现在也有些 RTC 内置了温度补偿，甚至还可以为系统提供环境温度值。

2. 常用的实时时钟芯片

我们见到最多的 RTC 可能是 DS1302 和 DS12887 了，当然还有很多其他的同类产品，按功能不同对几个比较常见的 RTC 予以简单的比较见表 10-16。

表 10-16　一些常用 RTC 的功能比较

RTC 型号	生产商	接口方式	晶振内置	补偿方式	温度补偿	电池内置	充电电路	报警输出
DS12887	DALLAS	并行	是	无	无	是	有	有
DS1302	DALLAS	串行	否	无	无	否	有	无
DS3231	DALLAS	串行	是	硬件	有	否	无	有
RX8025	EPSON	串行	是	软件	无	否	无	有
PCF8563	PHILIPS	串行	否	无	无	否	无	有

10.11.2　DS1302 的结构及工作原理

DS1302 是美国 DALLAS 公司推出的一种高性能、低功耗、带 RAM 的实时时钟芯片，它可以对年、月、周、日、时、分、秒进行计时，且具有闰年补偿功能，工作电压宽达 2.5 ~ 5.5V。采用三线串行接口与单片机进行同步通信，并可采用突发方式一次传送多个字节的时钟数据或 RAM 数据。同时 DS1302 内部还有一个 31 × 8 的用于临时存放数据的 RAM 寄存器。此外还有主电源/后备电源双电源引脚，并提供了对后备电源进行涓流充电的能力。DS1302 的引脚排列如图 10-87 所示。

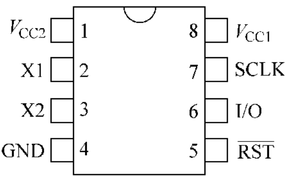


图 10-87　DS1302 的外部引脚排列

X1，X2—32.768kHz 晶振引脚　GND—地　RST—复位
I/O—数据输入/输出
SCLK—串行时钟
V_{CC1}—电池引脚
V_{CC2}—主电源引脚

1. DS1302 的结构

DS1302 的结构如图 10-88 所示，主要组成部分为：移位寄存器、控制逻辑、振荡器、实时时钟以及 RAM。虽然数据分成两种，但是对单片机的程序而言，其实是一样的，就是对特定

的地址进行读写操作。

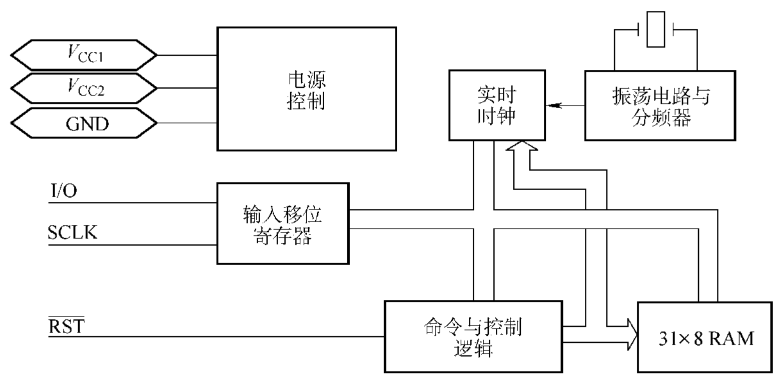


图 10-88　DS1302 的结构

2. DS1302 的工作原理

DS1302 含充电电路，可以对作为后备电源的可充电电池充电，并可选择充电使能和串入的二极管数目，以调节电池充电电压。不过对我们目前而言，最需要熟悉的是和时钟相关部分的功能，如有需要，请参阅数据手册。

实时时钟的实现，是通过计数器计数经过分频的振荡信号产生的，不过这些计数器被映射在地址空间里，这样，在时钟计时允许的前提下，我们就可以从表 10-17 给出的地址里读得时钟数据。我们先来看看跟时钟相关的寄存器分布在哪些位置上：

DS1302 内部共有 12 个寄存器，其中有 7 个寄存器与日历、时钟相关，即表 10-17 所列的数据，存放的数据位为 BCD 码形式。此外，DS1302 还有年份寄存器、控制寄存器、充电寄存器、时钟突发寄存器及与 RAM 相关的寄存器等。时钟突发寄存器可一次性顺序读写除充电寄存器以外的寄存器。

表 10-17　DS14302 内部跟时钟相关的寄存器分布

寄存器名称	命令字		取值范围	各位内容							
	写	读		7	6	5	4	3	2	1	0
秒寄存器	80H	81H	00～59	CH	10SEC			SEC			
分寄存器	82H	83H	00～59	0	10MIN			MIN			
小时寄存器	84H	85H	01～12 或 00～23	12/24	0	A	HR	HR			
日期寄存器	86H	87H	01～28，29，30，31	0	0	10DATE		DATE			
月份寄存器	88H	89H	01～12	0	0	0	10M	MONTH			
周寄存器	8AH	8BH	01～07	0	0	0	0	0	DAY		
年份寄存器	8CH	8CH	00～99	10YEAR				YEAR			

DS1302 内部的 RAM 分为两类，一类是单个 RAM 单元，共 31 个，每个单元组态为一个 8 位的字节，其命令控制字为 COH～FDH，其中奇数为读操作，偶数为写操作；再一类为突发方式下的 RAM，此方式下可一次性读写所有的 RAM 的 31 个字节，命令控制字为 FEH（写）、FFH（读）。

我们现在已经知道了控制寄存器和 RAM 的逻辑地址，接着就需要知道如何通过外部接口来访问这些资源。单片机是通过简单的同步串行通信与 DS1302 通信的，每次通信都必须

由单片机发起，无论是读还是写操作，单片机都必须先向 DS1302 写入一个命令帧，这个帧的格式如图 10-89 所示，最高位 bit7 固定为 1，bit6 决定操作是针对 RAM 还是时钟寄存器，接着的 5 个 bit 是 RAM 或时钟寄存器在 DS1302 的内部地址，最后一个 bit 表示这次操作是读操作抑或是写操作。

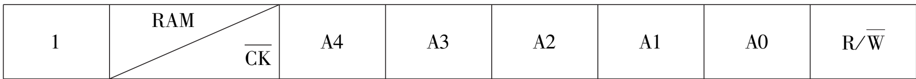


图 10-89　DS1302 的命令字结构

物理上，DS1302 的通信接口由 3 个口线组成，即 RST，SCLK，I/O。其中 RST 从低电平变成高电平启动一次数据传输过程，SCLK 是时钟线，I/O 是数据线。具体的读写时序如图 10-90 所示。但是请注意，无论是哪种同步通信类型的串行接口，都是对时钟信号敏感的，而且一般数据写入有效是在上升沿，读出有效是在下降沿（DS1302 正是如此的，但是在芯片手册里没有明确说明）。如果不是特别确定，则把程序设计成这样：平时 SCLK 保持低电平，在时钟变动前设置数据，在时钟变动后读取数据，即数据操作总是在 SCLK 保持为低电平的时候，相邻的操作之间间隔有一个上升沿和一个下降沿。

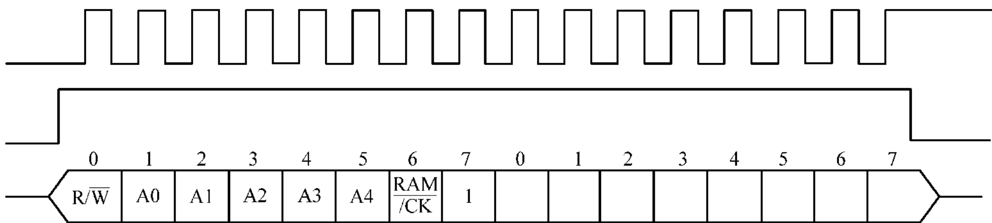


图 10-90　DS1302 的命令字结构

3. DS1302 和单片机之间的软件接口程序设计

其实单片机同外部器件的接口程序设计总是十分类似的，对通信口而言，最重要的就是时序，也就是我们在前面说到的数据读写的时刻问题。在设计这样的程序的时候，一定要十分留意这点。

由于接口程序设计的本身比较简单，请自行阅读第 3 小节的程序了解接口的实现。

10.11.3　DS1302 和单片机之间的接口程序实现

```
#include <reg52.h>
#include <intrins.h>

sbit DS1302_data   =  0xa1;
sbit DS1302_cs     =  0xa2;
sbit DS1302_clk    =  0xa0;
unsigned char display_buffer[6];

//延时程序
void delay()
```

```
{
    unsigned char i,j;
    for(i=0;i<200;i++)
        for(j=0;j<200;j++);
}

//短延时程序
void short_delay()
{
    unsigned char i;
    for(i=0;i<5;i++);
}

//长延时程序
void long_delay()
{
    unsigned char i;
    for(i=0;i<25;i++);
}

//向 DS1302 写入一个字节子程序
void DS1302_write(unsigned char DS1302_temp)
{
    unsigned char i,temp;
    long_delay();
    temp=DS1302_temp;
    for(i=0;i<8;i++){
        DS1302_data=temp&0x01;
        short_delay();
        DS1302_clk=1;
        short_delay();
        DS1302_clk=0;
        temp=temp>>1;
    }
}

//从 DS1302 读取一个字节子程序
unsigned char DS1302_read(void)
{
```

```
    unsigned char i,temp;
    long_delay();
    temp =0;
    for(i =0;i <8;i + +){
        temp = temp >>1;
        if(DS1302_data = =1) temp = temp +0x80;
        DS1302_clk =1;
        short_delay();
        DS1302_clk =0;
        short_delay();
    }
    return temp;
}

//主程序功能
//循环读取时钟数据到缓冲区 display_buffer[6]
void main(void)
{
    unsigned char i;
      DS1302_cs =0;

      while(1){

        DS1302_clk =0;
        short_delay();
        DS1302_cs =1;
        DS1302_write(0x81);
        DS1302_data =1;
        i =DS1302_read();
        DS1302_cs =0;
        short_delay();
        DS1302_clk =1;
        display_buffer[4] =(i >>4)&0x07;
        display_buffer[5] =i&0x0f;

        DS1302_clk =0;
        short_delay();
        DS1302_cs =1;
        DS1302_write(0x83);
```

```
        DS1302_data = 1;
        i = DS1302_read();
        DS1302_cs = 0;
        short_delay();
        DS1302_clk = 1;

        display_buffer[2] = (i >> 4)&0x07;
        display_buffer[3] = i&0x0f;

        DS1302_clk = 0;
        short_delay();
        DS1302_cs = 1;
        DS1302_write(0x85);
        DS1302_data = 1;
        i = DS1302_read();
        DS1302_cs = 0;
        short_delay();
        DS1302_clk = 1;

        delay();
    }
}
```

10.12　12864 点阵型 LCD 应用实例

在 10.2 节介绍了字符型 LCD 的基本应用，通过学习后，可以实现数字和字母的显示，但在现场应用中除了数字和字母外还要显示汉字。因为字符型 LCD 无法将汉字显示出来，所以在显示汉字的场合一般都用点阵型 LCD。目前，常用的点阵型 LCD 有 122×32、128×64、240×320 等。本节重点介绍 128×64 点阵显示屏的基本应用，128×64 点阵显示屏有三种控制器，分别是 KS0107（KS0108）、T6963C 和 ST7920。三种控制器主要区别是：KS0107（KS0108）不带任何字库，T6963C 带 ASCII 码，ST7920 带国标二级字库（8000 多个汉字）。本节以不带字库的 KS0107（KS0108）控制器 LCD 为例，介绍汉字的基本显示方法。

10.12.1　点阵型 LCD 的显示原理

在 10.2 节我们已经了解了 LCD 基本的显示原理，字符型和点阵型 LCD 的主要区别是：字符型 LCD 只能显示数字和字母符号，所需存储空间有限，所以一般都把基本字库表固化在自带的 ROM 里；而不带汉字库的点阵显示屏则不同，每个字符和汉字都要用户自己取模。下面简单介绍一下显示字符和显示汉字的区别。

在数字电路中，所有的数据都是以 0 和 1 保存的，对 LCD 控制器进行不同的数据操作，

可以得到不同的结果。对于显示英文操作，由于英文字母种类很少，只需要 8 位（一字节）即可。而对于中文，常用的字却有 6000 个以上，于是我们的 DOS 前辈想了一个办法，就是将 ASCII 表的高 128 个很少用到的数值以两个为一组来表示汉字，即汉字的内码。而剩下的低 128 位则留给英文字符使用，即英文的内码。

那么，得到了汉字的内码后，还仅是一组数字，那又如何在屏幕上去显示呢？这就涉及到文字的字模，字模虽然也是一组数字，但它的意义却与数字的意义有了根本的变化，它是用数字的各位信息来记载英文或汉字的形状，如英文的“A”在字模的记载方式如图 10-91 所示。

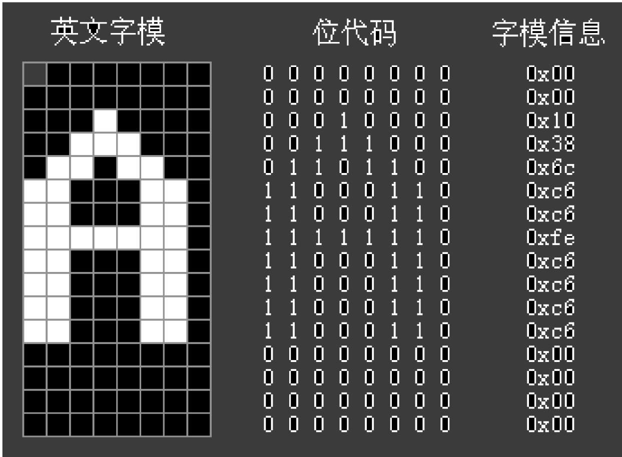


图 10-91　“A”字模图

而中文的“你”在字模中的记载却如图 10-92 所示。

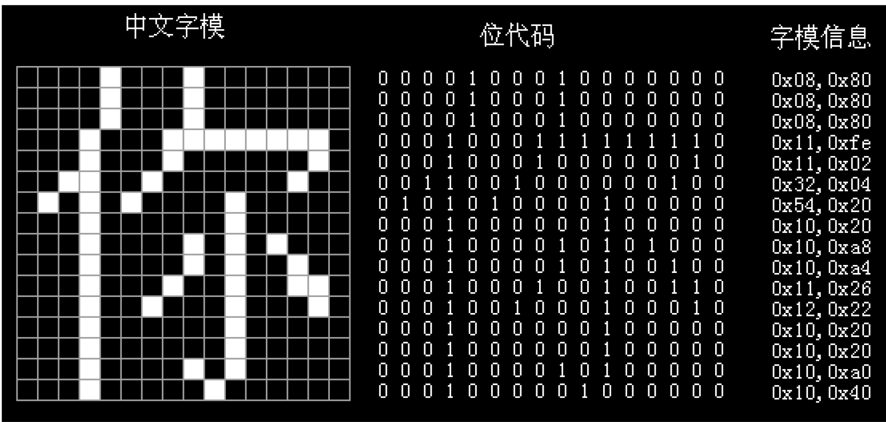


图 10-92　“你”字模图

10.12.2　12864 点阵型 LCD 简介

12864 是一种图形点阵液晶显示器，它主要由行驱动器/列驱动器及 128 × 64 全点阵液晶显示器组成。可完成图形显示，也可以显示 8 × 4 个（16 × 16 点阵）汉字。

主要技术参数和性能：

1）电源：VDD +5V；模块内自带 -10V 负电压，用于 LCD 的驱动电压；
2）显示内容：128（列）×64（行）点；
3）全屏幕点阵；
4）七种指令；
5）与 CPU 接口采用 8 位数据总线并行输入输出和 8 条控制线；
6）占空比 1/64；
7）工作温度：-20℃ ~ +60℃，存储温度：-30℃ ~ +70℃。

1. 12864LCD 的外形结构及引脚说明

其外形尺寸如图 10-93 所示，引脚说明见表 10-18。

表 10-18　12864LCD 引脚说明

引脚号	引脚名称	电　　平	引脚功能描述
1	VSS	0	电源地
2	VDD	+5.0V	电源电压
3	V0	-	液晶显示器驱动电压
4	D/I（RS）	H/L	D/I =“H”，表示 DB7 ~ DB0 为显示数据 D/I =“L”，表示 DB7 ~ DB0 为显示指令数据
5	R/W	H/L	R/W =“H”，E =“H”数据被读到 DB7 ~ DB0 R/W =“L”，E =“H→L”数据被写到 IR 或 DR
6	E	H/L	R/W =“L”，E 信号下降沿锁存 DB7 ~ DB0 R/W =“H”，E =“H”DDRAM 数据读到 DB7 ~ DB0
7	DB0	H/L	数据线
8	DB1	H/L	数据线
9	DB2	H/L	数据线
10	DB3	H/L	数据线
11	DB4	H/L	数据线
12	DB5	H/L	数据线
13	DB6	H/L	数据线
14	DB7	H/L	数据线
15	CS1	H/L	H：选择芯片（右半屏）信号
16	CS2	H/L	H：选择芯片（左半屏）信号
17	RET	H/L	复位信号，低电平复位
18	VOUT	-10V	LCD 驱动负电压
19	LED +	—	LED 背光板电源
20	LED -	—	LED 背光板电源

2. 12864LCD 的内部模块结构

本节所介绍的液晶显示模块均为 KS0108B 及其兼容控制驱动器（例如 HD61202）作为列驱动器，同时使用 KS0107B 及其兼容驱动器（例如 HD61203）作为行驱动器的液晶模块。由于 KS0107B（或 HD61203）不与 MPU 发生联系，只要提供电源就能产生行驱动信号和各种同步信号，比较简单，在此就不作介绍。图 10-94 是 12864 的逻辑电路图，12864 共有两片 KS0108B 或兼容控制驱动器和一片 HD61203 或兼容驱动器。

从图 10-94 可以看出，12864 的 LCD 基本由数据线（DB0 ~ DB7）和控制线组成，左右半屏的显示控制由 CS1、CS2 控制，CS1 和 CS2 不同的组合可以控制屏幕左右显示。12864LCD 内部结构如图 10-95 所示。

在使用 12864LCD 前先必须了解以下功能器件才能进行编程。12864 内部功能器件及相

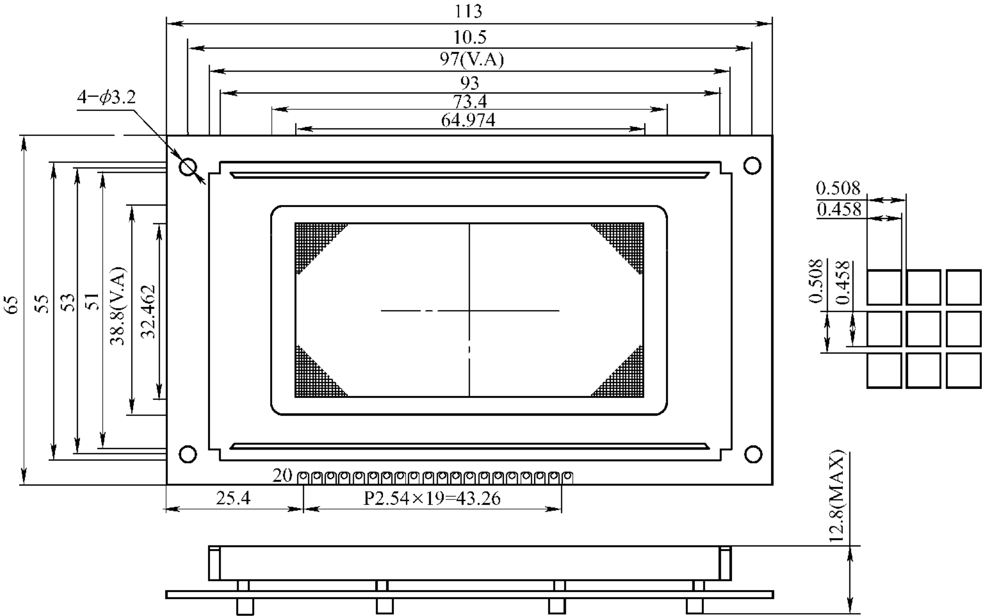


图 10-93　外形尺寸图

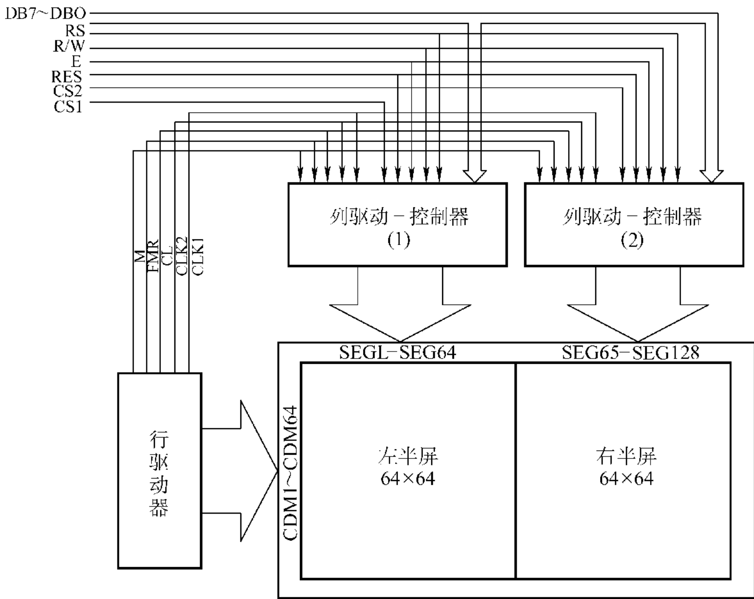


图 10-94　内部逻辑电路图

关功能如下：

（1）指令寄存器（IR）

IR 是用于寄存指令码的，与数据寄存器数据相对应。当 D/I =0 时，在 E 信号下降沿的作用下，指令码写入 IR。

（2）数据寄存器（DR）

DR 是用于寄存数据的，与指令寄存器寄存指令相对应。当 D/I =1 时，在下降沿作用

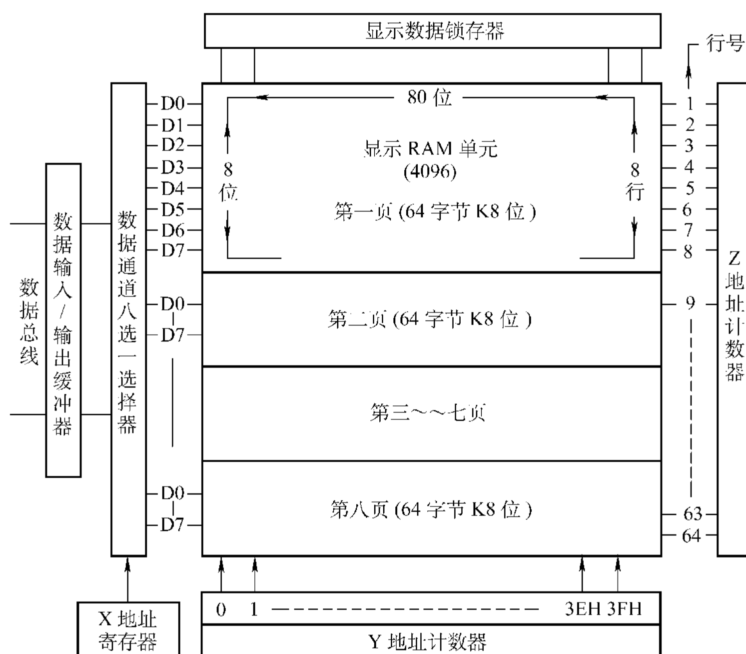


图 10-95 内部结构

下，图形显示数据写入 DR，或在 E 信号高电平作用下由 DR 读到 DB7～DB0 数据总线。DR 和 DDRAM 之间的数据传输是模块内部自动执行的。

（3）忙标志（BF）

BF 标志提供内部工作情况。BF＝1 表示模块在内部操作，此时模块不接受外部指令和数据。BF＝0 时，模块为准备状态，随时可接受外部指令和数据。

利用 STATUS READ 指令，可以将 BF 读到 DB7 总线，检验模块之工作状态。

（4）显示控制触发器（DFF）

此触发器是用于模块屏幕显示开和关的控制。DFF＝1 为开显示（DISPLAY OFF），DDRAM 的内容就显示在屏幕上，DFF＝0 为关显示（DISPLAY OFF）。

DDF 的状态是指令 DISPLAY ON/OFF 和 RST 信号控制的。

（5）XY 地址计数器

XY 地址计数器是一个 9 位计数器。高 3 位是 X 地址计数器，低 6 位为 Y 地址计数器，XY 地址计数器实际上是作为 DDRAM 的地址指针，X 地址计数器为 DDRAM 的页指针，Y 地址计数器为 DDRAM 的 Y 地址指针。

X 地址计数器是没有记数功能的，只能用指令设置。

Y 地址计数器具有循环记数功能，各显示数据写入后，Y 地址自动加 1，Y 地址指针从 0～63。

（6）显示数据 RAM（DDRAM）

DDRAM 是存储图形显示数据的。数据为 1 表示显示选择，数据为 0 表示显示非选择。DDRAM 与地址和显示位置的关系见 DDRAM 地址表。

（7）Z 地址计数器

Z 地址计数器是一个 6 位计数器，此计数器具备循环记数功能，用于显示行扫描同步。

当一行扫描完成，此地址计数器自动加 1，指向下一行扫描数据，RST 复位后 Z 地址计数器为 0。

Z 地址计数器可以用指令 DISPLAY START LINE 预置。因此，显示屏幕的起始行就由此指令控制，即 DDRAM 的数据从哪一行开始显示在屏幕的第一行。此模块的 DDRAM 共 64 行，屏幕可以循环滚动显示 64 行。

3. 12864LCD 的指令系统及时序

该类液晶显示模块（即 KS0108B 及其兼容控制驱动器）的指令系统比较简单，总共只有七种。其指令见表 10-19。

表 10-19　12864LCD 指令表

指令名称	控制信号		控制代码							
	R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
显示开关	0	0	0	0	1	1	1	1	1	1/0
显示起始行设置	0	0	1	1	X	X	X	X	X	X
页设置	0	0	1	0	1	1	1	X	X	X
列地址设置	0	0	0	1	X	X	X	X	X	X
读状态	1	0	BUSY	0	ON/OFF	RST	0	0	0	0
写数据	0	1	写数据							
读数据	1	1	读数据							

各功能指令分别介绍如下。

（1）显示开/关指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	1/0

当 DB0 = 1 时，LCD 显示 RAM 中的内容；DB0 = 0 时，关闭显示。

（2）显示起始行（ROW）设置指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	显示起始行（0～63）					

该指令设置了对应液晶屏最上一行的显示 RAM 的行号，有规律地改变显示起始行，可以使 LCD 实现显示滚屏的效果。

（3）页（PAGE）设置指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	页号（0～7）		

显示 RAM 共 64 行，分 8 页，每页 8 行。

（4）列地址（Y Address）设置指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	显示列地址（0～63）					

设置了页地址和列地址，就惟一确定了显示 RAM 中的一个单元，这样 MPU 就可以用读、写指令读出该单元中的内容或向该单元写进一个字节数据。

（5）读状态指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	BUSY	0	ON/OFF	REST	0	0	0	0

该指令用来查询液晶显示模块内部控制器的状态，各参量含义如下：

BUSY　：　1-内部在工作　　0-正常状态

ON/OFF　：　1-显示关闭　　0-显示打开

RESET　：　1-复位状态　　0-正常状态

在 BUSY 和 RESET 状态时，除读状态指令外，其他指令均不对液晶显示模块产生作用。

在对液晶显示模块操作之前要查询 BUSY 状态，以确定是否可以对液晶显示模块进行操作。

（6）写数据指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	写　数　据							

（7）读数据指令

R/W	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	读显示数据							

读、写数据指令每执行完一次读、写操作，列地址就自动增一。必须注意的是，进行读操作之前，必须有一次空读操作，紧接着再读才会读出所要读的单元中的数据。

LCD 基本读/写操作时序如图 10-96 所示。

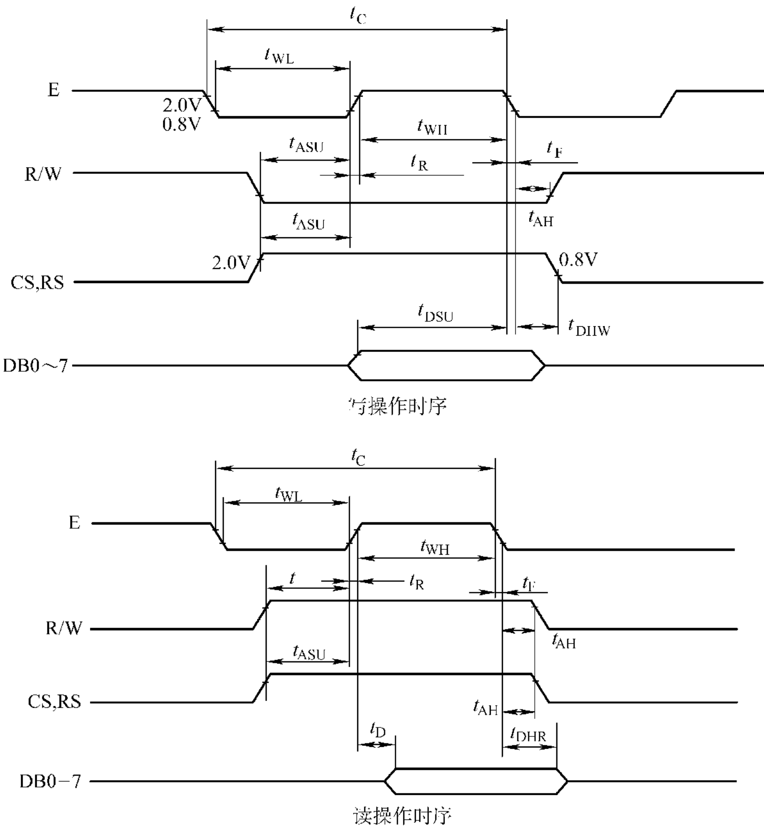
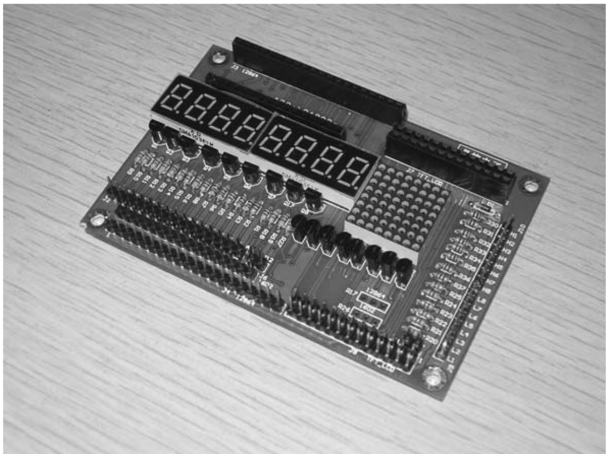


图 10-96　读/写操作时序

10.12.3　12864 点阵型 LCD 软硬件设计实例

通过以上的学习，现在就来实际应用 12864LCD 的软硬件设计，本实验可以直接在配套开发板与 KC-102 单片机显示板模块连接完成，如图 10-97a。本实例将在 LCD 上显示如图 10-98 所示。



a)



b)

图 10-97　128×64LCD 实验演示图

a）KC-103 单片机键盘板模块　b）128×64LCD 实验演示

				欢		迎		使		用					
		单		片		机		开		发		板			
当		前		状		态		:	运		行		中		…
w	w	w	.	h	i	f	i	c	a	t	.	c	o	m	

图 10-98　模拟显示效果图

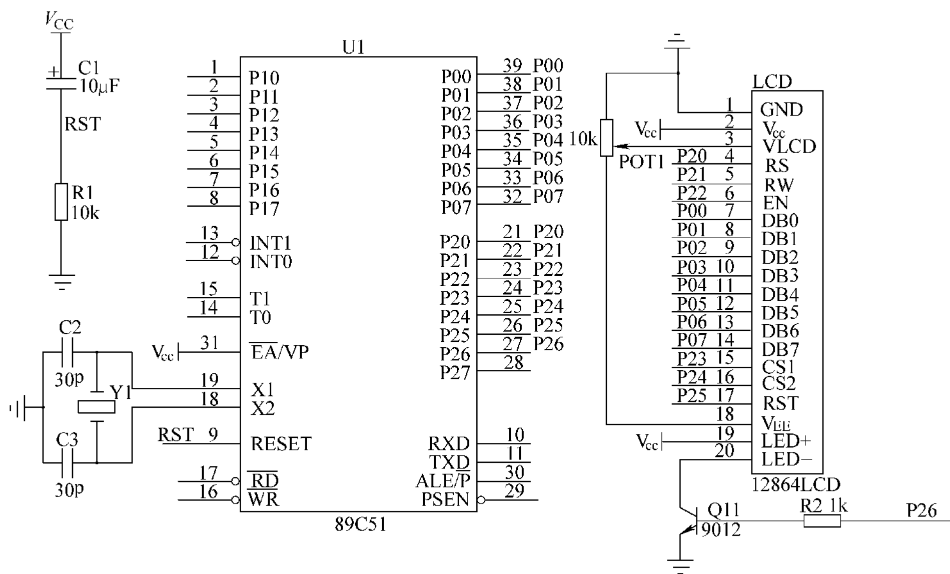


图 10-99　硬件原理图

1. 硬件原理图

如图 10-99 所示。

2. 程序流程图

如图 10-100 所示。

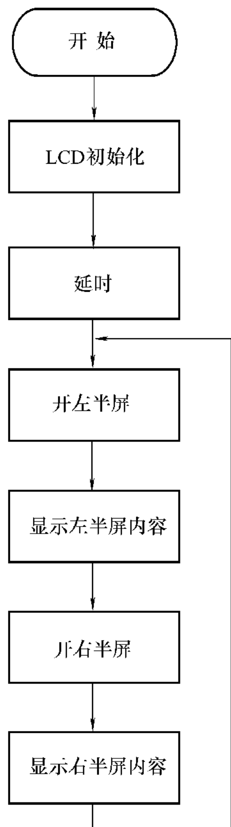


图 10-100　软件流程图

3. 软件代码

在编写软件代码之前必须要先掌握汉字取模的方法。要得到上表中的文字，我们可以借助取模软件来完成。目前点阵 LCD 的取模软件有很多，我们以本开发板配套的取模软件为例来介绍一下汉字的取模方法。

打开取模软件出现显示界面，如图 10-101 所示。

在文字输入区中输入文字，我们以输入一个欢迎的“欢”字为例，了解其取模过程。在文字输入区中输入“欢”后按“CTRL + ENTER”组合键后就看到“欢”字已经在模拟显示区显示出来了，如图 10-102 所示。

在“取模方式”中选择“C51 格式”就可以在“点阵生成区”得到你要的汉字

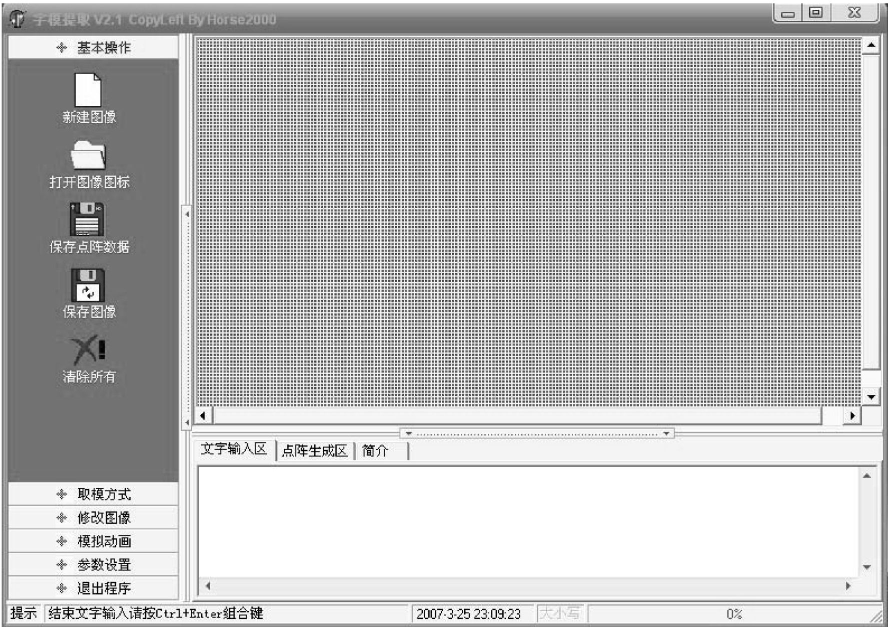


图 10-101　字模提取界面

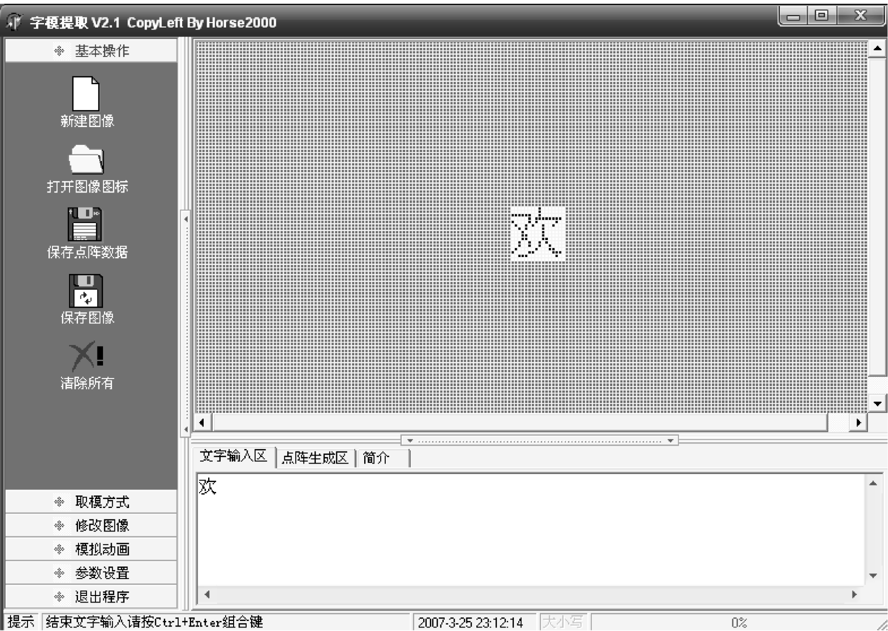


图 10-102　取模过程

“欢”的显示代码。



文字输入区　点阵生成区　简介

```
/*--  文字:  欢  --*/
/*--  宋体12;  此字体下对应的点阵为：宽x高=16x16   --*/
0x00,0x80,0x00,0x80,0xFC,0x80,0x05,0xFE,0x85,0x04,0x4A,0x48,0x28,0x40,0x10,0x40,
0x18,0x40,0x18,0x60,0x24,0xA0,0x24,0x90,0x41,0x18,0x86,0x0E,0x38,0x04,0x00,0x00
```

经过以上步骤后一个汉字就取模成功了，在程序中只要调用这段代码就可显示出汉字“欢”了，其他汉字也用同样的方法。取完要显示的全部汉字代码后我们就可以编程了，最终的软件代码如下：

```
/*****************************************/
/*杭州晶控电子有限公司                                */
/*http://www. hificat. com                            */
/*12864LCD 演示程序                                   */
/*目标器件：AT89S51                                   */
/*晶振：11.0592MHz                                    */
/*编译环境：Keil 7.50A                                */
/*****************************************/
/*******************包含头文件*********************/
#include <reg51.h>

/*******************命令字定义*********************/
#define Disp_On 0x3f
#define Disp_Off 0x3e
#define Col_Add 0x40
#define Page_Add 0xb8
#define Start_Line 0xc0

/******************** 端口定义*********************/
sbit Mcs = P2^3;                     //左半屏使能，当 MCS=1，左半屏显示
sbit Scs = P2^4;                     //右半屏使能，当 SCS=1，右半屏显示
sbit Enable = P2^2;                  //使能
sbit Di = P2^0;                      //数据/命令选择（RS）
sbit RW = P2^1;                      //读/写信号
sbit Rst = P2^5;                     //复位脚
sbit Light = P2^6;                   //背光脚

/ **********************字模表*********************/
/******************www.hificat.com*******************/
char code h [ ] = {
/* -- 文字：h -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 =8x16 -- */
0x08, 0xF8, 0x00, 0x80, 0x80, 0x80, 0x00, 0x00, 0x20, 0x3F, 0x21, 0x00, 0x00,
0x20, 0x3F, 0x20,};
char code w [ ] = {
/* -- 文字：w -- */
```

```
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x80, 0x80, 0x00, 0x80, 0x00, 0x80, 0x80, 0x80, 0x0F, 0x30, 0x0C, 0x03, 0x0C,
0x30, 0x0F, 0x00,};
char code i [] = {
/* -- 文字: i -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x80, 0x98, 0x98, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0x3F, 0x20,
0x20, 0x00, 0x00,};
char code f [] = {
/* -- 文字: f -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x80, 0x80, 0xF0, 0x88, 0x88, 0x88, 0x18, 0x00, 0x20, 0x20, 0x3F, 0x20,
0x20, 0x00, 0x00,};
char code c [] = {
/* -- 文字: c -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x0E, 0x11, 0x20, 0x20,
0x20, 0x11, 0x00,};
char code a [] = {
/* -- 文字: a -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x19, 0x24, 0x22, 0x22,
0x22, 0x3F, 0x20,};
char code t [] = {
/* -- 文字: t -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x80, 0x80, 0xE0, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x20,
0x20, 0x00, 0x00,};
char code o [] = {
/* -- 文字: o -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x1F, 0x20, 0x20, 0x20,
0x20, 0x1F, 0x00,};
char code m [] = {
/* -- 文字: m -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x20, 0x3F, 0x20, 0x00, 0x3F,
0x20, 0x00, 0x3F,};
char code dian [] = {
```

```
/* -- 文字: . -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =8x16 -- */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x00, 0x00,
0x00, 0x00, 0x00,};
/********************欢迎使用**********************/
char code huan [] = {
/* -- 文字: 欢 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
0x14, 0x24, 0x44, 0x84, 0x64, 0x1C, 0x20, 0x18, 0x0F, 0xE8, 0x08, 0x08, 0x28,
0x18, 0x08, 0x00, 0x20, 0x10, 0x4C, 0x43, 0x43, 0x2C, 0x20, 0x10, 0x0C, 0x03,
0x06, 0x18, 0x30, 0x60, 0x20, 0x00,};
char code yun2 [] = {
/* -- 文字: 迎 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
0x40, 0x41, 0xCE, 0x04, 0x00, 0xFC, 0x04, 0x02, 0x02, 0xFC, 0x04, 0x04, 0x04,
0xFC, 0x00, 0x00, 0x40, 0x20, 0x1F, 0x20, 0x40, 0x47, 0x42, 0x41, 0x40, 0x5F,
0x40, 0x42, 0x44, 0x43, 0x40, 0x00,};
char code shi [] = {
/* -- 文字: 使 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
0x40, 0x20, 0xF0, 0x1C, 0x07, 0xF2, 0x94, 0x94, 0x94, 0xFF, 0x94, 0x94, 0x94,
0xF4, 0x04, 0x00, 0x00, 0x00, 0x7F, 0x00, 0x40, 0x41, 0x22, 0x14, 0x0C, 0x13,
0x10, 0x30, 0x20, 0x61, 0x20, 0x00,};
char code yong [] = {
/* -- 文字: 用 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
0x00, 0x00, 0x00, 0xFE, 0x22, 0x22, 0x22, 0x22, 0xFE, 0x22, 0x22, 0x22, 0x22,
0xFE, 0x00, 0x00, 0x80, 0x40, 0x30, 0x0F, 0x02, 0x02, 0x02, 0x02, 0xFF, 0x02,
0x02, 0x42, 0x82, 0x7F, 0x00, 0x00,};
/ *******************单片机开发板*******************/
char code dan [] = {
/* -- 文字: 单 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
0x00, 0x00, 0xF8, 0x28, 0x29, 0x2E, 0x2A, 0xF8, 0x28, 0x2C, 0x2B, 0x2A, 0xF8,
0x00, 0x00, 0x00, 0x08, 0x08, 0x0B, 0x09, 0x09, 0x09, 0x09, 0xFF, 0x09, 0x09,
0x09, 0x09, 0x0B, 0x08, 0x08, 0x00,};
char code pian [] = {
//* -- 文字: 片 -- */
/* -- 宋体 12; 此字体下对应的点阵为: 宽 x 高 =16x16 -- */
```

```
0x00, 0x00, 0x00, 0xFE, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F, 0x10, 0x10, 0x10,
0x18, 0x10, 0x00, 0x80, 0x40, 0x30, 0x0F, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0xFF, 0x00, 0x00, 0x00, 0x00,};
char code ji [ ] = {
/* -- 文字：机 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x08, 0x08, 0xC8, 0xFF, 0x48, 0x88, 0x08, 0x00, 0xFE, 0x02, 0x02, 0x02, 0xFE,
0x00, 0x00, 0x00, 0x04, 0x03, 0x00, 0xFF, 0x00, 0x41, 0x30, 0x0C, 0x03, 0x00,
0x00, 0x00, 0x3F, 0x40, 0x78, 0x00,};
char code kai [ ] = {
/* -- 文字：开 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x40, 0x42, 0x42, 0x42, 0x42, 0xFE, 0x42, 0x42, 0x42, 0x42, 0xFE, 0x42, 0x42,
0x42, 0x42, 0x00, 0x00, 0x40, 0x20, 0x10, 0x0C, 0x03, 0x00, 0x00, 0x00, 0x00,
0x7F, 0x00, 0x00, 0x00, 0x00, 0x00,};
char code fa [ ] = {
/* -- 文字：发 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x00, 0x10, 0x3E, 0x10, 0x10, 0xF0, 0x9F, 0x90, 0x90, 0x92, 0x94, 0x1C, 0x10,
0x10, 0x10, 0x00, 0x40, 0x20, 0x10, 0x88, 0x87, 0x41, 0x46, 0x28, 0x10, 0x28,
0x27, 0x40, 0xC0, 0x40, 0x00, 0x00,};
char code ban [ ] = {
/* -- 文字：板 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x10, 0x10, 0xD0, 0xFF, 0x50, 0x90, 0x00, 0xFE, 0x62, 0xA2, 0x22, 0x21, 0xA1,
0x61, 0x00, 0x00, 0x04, 0x03, 0x00, 0x7F, 0x00, 0x11, 0x0E, 0x41, 0x20, 0x11,
0x0A, 0x0E, 0x31, 0x60, 0x20, 0x00,};

char code dang [ ] = {
/* -- 文字：当 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x00, 0x00, 0x40, 0x42, 0x5C, 0x48, 0x40, 0x40, 0x7F, 0x40, 0x50, 0x4E, 0x44,
0xC0, 0x00, 0x00, 0x00, 0x00, 0x20, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22,
0x22, 0x22, 0x22, 0x7F, 0x00, 0x00,};

char code qian [ ] = {
/* -- 文字：前 -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 = 16x16 -- */
0x08, 0x08, 0xE8, 0xA8, 0xA9, 0xAE, 0xEA, 0x08, 0x08, 0xC8, 0x0C, 0x0B, 0xEA,
```

```
0x08, 0x08, 0x00, 0x00, 0x00, 0x7F, 0x04, 0x24, 0x44, 0x3F, 0x00, 0x00, 0x1F,
0x40, 0x80, 0x7F, 0x00, 0x00, 0x00,};

char code zhuang [ ] = {
/* -- 文字:状 -- */
/* -- 宋体12;此字体下对应的点阵为:宽x高=16x16 -- */
0x08, 0x30, 0x00, 0xFF, 0x20, 0x20, 0x20, 0x20, 0xFF, 0x20, 0xE1, 0x26, 0x2C,
0x20, 0x20, 0x00, 0x04, 0x02, 0x01, 0xFF, 0x40, 0x20, 0x18, 0x07, 0x00, 0x00,
0x03, 0x0C, 0x30, 0x60, 0x20, 0x00,};

char code tai1 [ ] = {
/* -- 文字:态 -- */
/* -- 宋体12;此字体下对应的点阵为:宽x高=16x16 -- */
0x00, 0x04, 0x04, 0x04, 0x84, 0x44, 0x34, 0x4F, 0x94, 0x24, 0x44, 0x84, 0x84,
0x04, 0x00, 0x00, 0x00, 0x60, 0x39, 0x01, 0x00, 0x3C, 0x40, 0x42, 0x4C, 0x40,
0x40, 0x70, 0x04, 0x09, 0x31, 0x00,};

char code yun [ ] = {
/* -- 文字:运 -- */
/* -- 宋体12;此字体下对应的点阵为:宽x高=16x16 -- */
0x40, 0x41, 0xCE, 0x04, 0x00, 0x20, 0x22, 0xA2, 0x62, 0x22, 0xA2, 0x22, 0x22,
0x22, 0x20, 0x00, 0x40, 0x20, 0x1F, 0x20, 0x28, 0x4C, 0x4A, 0x49, 0x48, 0x4C,
0x44, 0x45, 0x5E, 0x4C, 0x40, 0x00,};

char code xing [ ] = {
/* -- 文字:行 -- */
/* -- 宋体12;此字体下对应的点阵为:宽x高=16x16 -- */
0x10, 0x08, 0x84, 0xC6, 0x73, 0x22, 0x40, 0x44, 0x44, 0x44, 0xC4, 0x44, 0x44,
0x44, 0x40, 0x00, 0x02, 0x01, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x40, 0x80,
0x7F, 0x00, 0x00, 0x00, 0x00, 0x00,};

char code zhong [ ] = {
/* -- 文字:中 -- */
/* -- 宋体12;此字体下对应的点阵为:宽x高=16x16 -- */
0x00, 0x00, 0xFC, 0x08, 0x08, 0x08, 0x08, 0xFF, 0x08, 0x08, 0x08, 0x08, 0xFC,
0x08, 0x00, 0x00, 0x00, 0x00, 0x07, 0x02, 0x02, 0x02, 0x02, 0xFF, 0x02, 0x02,
0x02, 0x02, 0x07, 0x00, 0x00, 0x00,};

char code maohao [ ] = {
```

```
/* -- 文字：：  -- */
/* -- 宋体 12；此字体下对应的点阵为：宽 x 高 =8x16  -- */
0x00，0x00，0x00，0xC0，0xC0，0x00，0x00，0x00，0x00，0x00，0x00，0x30，0x30，
0x00，0x00，0x00，}；

/****************************************************
函数功能：延时程序
入口参数：t
出口参数：
****************************************************/
void delay（unsigned int t）
{
    unsigned int i，j；
    for（i=0；i<t；i++）
    for（j=0；j<10；j++）；
}

/****************************************************
函数功能：写命令到 LCD 程序
入口参数：cmdcode
出口参数：
****************************************************/
void write_com（unsigned char cmdcode）
{
    Di=0；
    RW=0；
    P0=cmdcode；
    delay（2）；
    Enable=1；
    delay（2）；
    Enable=0；
}

/****************************************************
函数功能：写数据到 LCD 程序
入口参数：Dispdata
出口参数：
****************************************************/
void write_data（unsigned char Dispdata）
```

```
{
    Di =1;
    RW =0;
    P0 = Dispdata;
    delay (2);
    Enable =1;
    delay (2);
    Enable =0;
}

/****************************************************
函数功能：清除 LCD 内存程序
入口参数：pag，col，hzk
出口参数：
****************************************************/
void Clr_Scr ( )
{
    unsigned char j, k;
    Mcs =1; Scs =1;
    write_com (Page_Add +0);
    write_com (Col_Add +0);
    for (k =0; k <8; k + + )
        {
        write_com (Page_Add + k);
        for (j =0; j <64; j + + ) write_data (0x00);
        }
    Mcs =0; Scs =0;
}

/****************************************************
函数功能：指定位置显示数字 16 * 16 程序
入口参数：pag，col，hzk
出口参数：
****************************************************/
void hz_disp16 (unsigned char pag, unsigned char col, unsigned char code *hzk)
{
    unsigned char j =0, i =0;
    for (j =0; j <2; j + + )
      {
```

```
            write_com (Page_Add + pag + j);
            write_com (Col_Add + col);
            for (i = 0; i < 16; i + +)
            write_data (hzk [16 * j + i]);
        }
}

/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能：指定位置显示数字 8 * 16 程序
入口参数：pag, col, hzk
出口参数：
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
void hz_disp8 (unsigned char pag, unsigned char col, unsigned char code * hzk)
{
    unsigned char j = 0, i = 0;
    for (j = 0; j < 2; j + +)
      {
            write_com (Page_Add + pag + j);
            write_com (Col_Add + col);
            for (i = 0; i < 8; i + +)
            write_data (hzk [8 * j + i]);
        }
}

/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
函数功能：LCD 初始化程序
入口参数：
出口参数：
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
void init_lcd ()
{
    delay (100);
    Mcs = 1;
    Scs = 1;
    delay (100);
    write_com (Disp_Off);
    write_com (Page_Add + 0);
    write_com (Start_Line + 0);
    write_com (Col_Add + 0);
```

```
    write_com (Disp_On);
}

/*****************************************
函数功能：主程序
入口参数：
出口参数：
*****************************************/
void main (void)
{
    Light =0;                       //开 LCD 背光
    init_lcd ( );
    Clr_Scr ( );
    Mcs =1; Scs =0;                 //左、右都显示
    while (1)
     {
        Mcs =1; Scs =0;             //左显示
        delay (2);
        //欢迎
        hz_disp16 (0, 32, huan);
        hz_disp16 (0, 48, yun2);
        //单片机
        hz_disp16 (2, 16, dan);
        hz_disp16 (2, 32, pian);
        hz_disp16 (2, 48, ji);
        //当前状态
        hz_disp16 (4, 0, dang);
        hz_disp16 (4, 16, qian);
        hz_disp16 (4, 32, zhuang);
        hz_disp16 (4, 48, tai1);
        //网址：www. hifi
        hz_disp8 (6, 0, w);
        hz_disp8 (6, 8, w);
        hz_disp8 (6, 16, w);
        hz_disp8 (6, 24, dian);
        hz_disp8 (6, 32, h);
        hz_disp8 (6, 40, i);
        hz_disp8 (6, 48, f);
        hz_disp8 (6, 56, i);
```

```
        Mcs = 0; Scs = 1;                    //右显示
        //使用
        hz_disp16 (0, 0, shi);
        hz_disp16 (0, 16, yong);
        //开发板
        hz_disp16 (2, 0, kai);
        hz_disp16 (2, 16, fa);
        hz_disp16 (2, 32, ban);
        //: 运行中
        hz_disp8 (4, 0, maohao);
        hz_disp16 (4, 8, yun);
        hz_disp16 (4, 24, xing);
        hz_disp16 (4, 40, zhong);
        //网址: cat. com
        hz_disp8 (6, 0, c);
        hz_disp8 (6, 8, a);
        hz_disp8 (6, 16, t);
        hz_disp8 (6, 24, dian);
        hz_disp8 (6, 32, c);
        hz_disp8 (6, 40, o);
        hz_disp8 (6, 48, m);
        delay (2);
    }
}
```

第 11 章　新型单片机外扩展模块

11.1　KC-101 51/AVR 单片机最小系统核心板

1. 功能描述

KC-101 为 51 单片机最小系统核心板，它包括了 51 单片机的复位电路、振荡电路所组成的最小系统单元，同时提供了单片机 32 个 P 口的 LED 状态指示灯，使用户在学习和调试程序时能够直观地、实时地观察各 P 口的电平状态，非常实用而有意思。板子虽小，但接口非常齐全，电源可以使用 USB 供电和外接电源供电两种方式，通过板子上的优质按键开关来切换，而不是像市场上有些产品用跳线那种麻烦的方式。

KC-101 51/AVR 单片机最小系统核心板外观如图 11-1 所示。

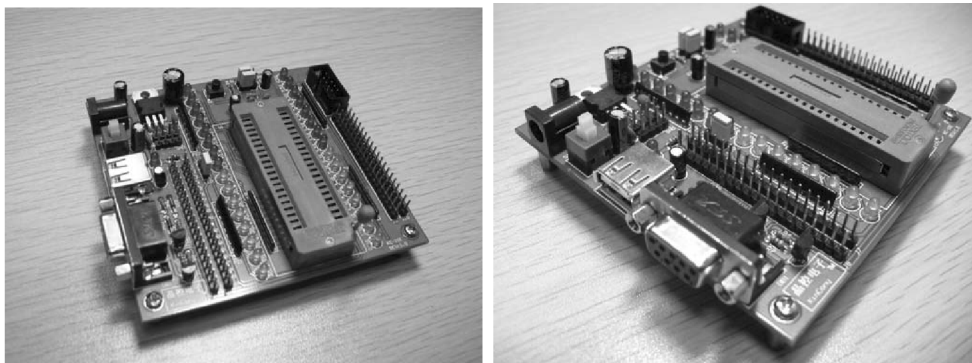


图 11-1　KC-101 51/AVR 单片机系统核心板

2. 硬件资源描述

1）32 路 LED 流水灯：可以显示 P 口的状态，不同的 P 口用不同的颜色来表示，显示更为直观。可做的实验如：正反流水灯、交通指示等等。

2）含有 RS232 接口，支持串口通信实验。

3）所有芯片引脚都接有外扩排针，我们提供了双排排针，为系统扩展做了更大的余地和空间，没有限制外扩实验的功能，完全由用户决定，同时提供了丰富的 VCC 和 GND 扩展接口。

4）支持 STC 单片机 ISP 在线下载功能，通过串口烧写芯片，非常方便。

5）板载标准 10PIN ISP 下载线接线，支持烧写 AT89S 系列单片机芯片。

6）支持使用 AVR ATMEGA8515/ATMEGA8515L 单片机芯片，由于 AVR 和 51 单片机复位电路不同。板上已设有跳线，更改跳线即可适用于不同芯片。

7）支持外范围的外接电源输入，如果使用外接电源，直流 9～12V 的电源都可以适用。

8）使用优质的 PCB 板材，双面板铺铜走线设计，抗干扰性好。

3. 硬件接口描述

J1：外接电源口：输入直流 9～12V 电压。

S1：电源开关：用来切换 USB 供电或外接电源供电。

J0：USB 供电接口：通过 USB 线缆向 PC 取电。

DB1：RS232 接口：完成串口通信及部分单片机芯片的烧写工作。

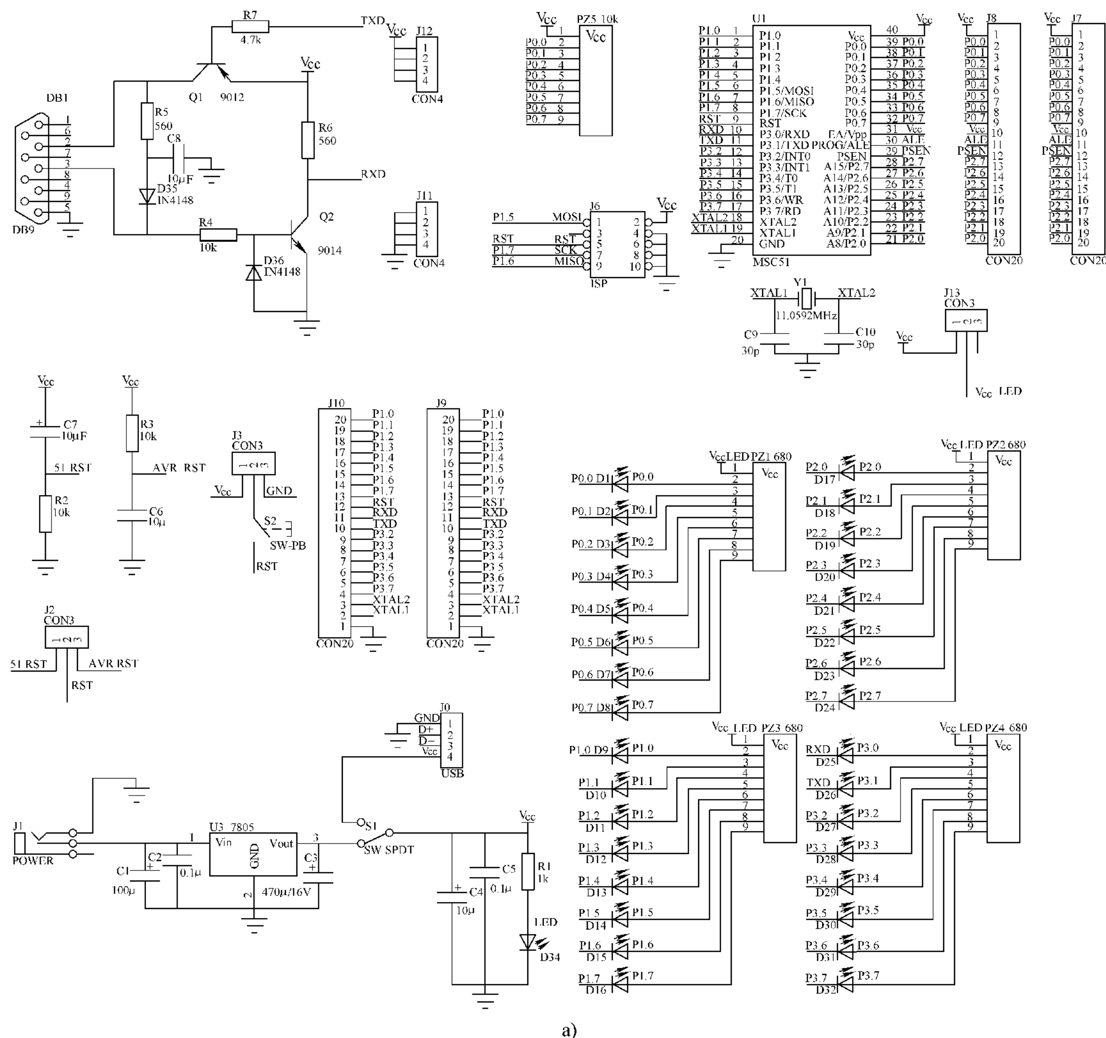
J7、J8、J9、J10：分别为 51 单片机所有引脚的外扩展接口。

J13：32 路 LED 发光管的使能跳线，可以使能 32 路发光管或屏蔽 32 路发光管，让 P 口完全独立。

S2：单片机的复位按键。

J2、J3：复位类型选择：用来切换 51/AVR 单片机的复位模式，左侧为 51 单片机复位方式；右侧为 AVR 单片机复位方式。

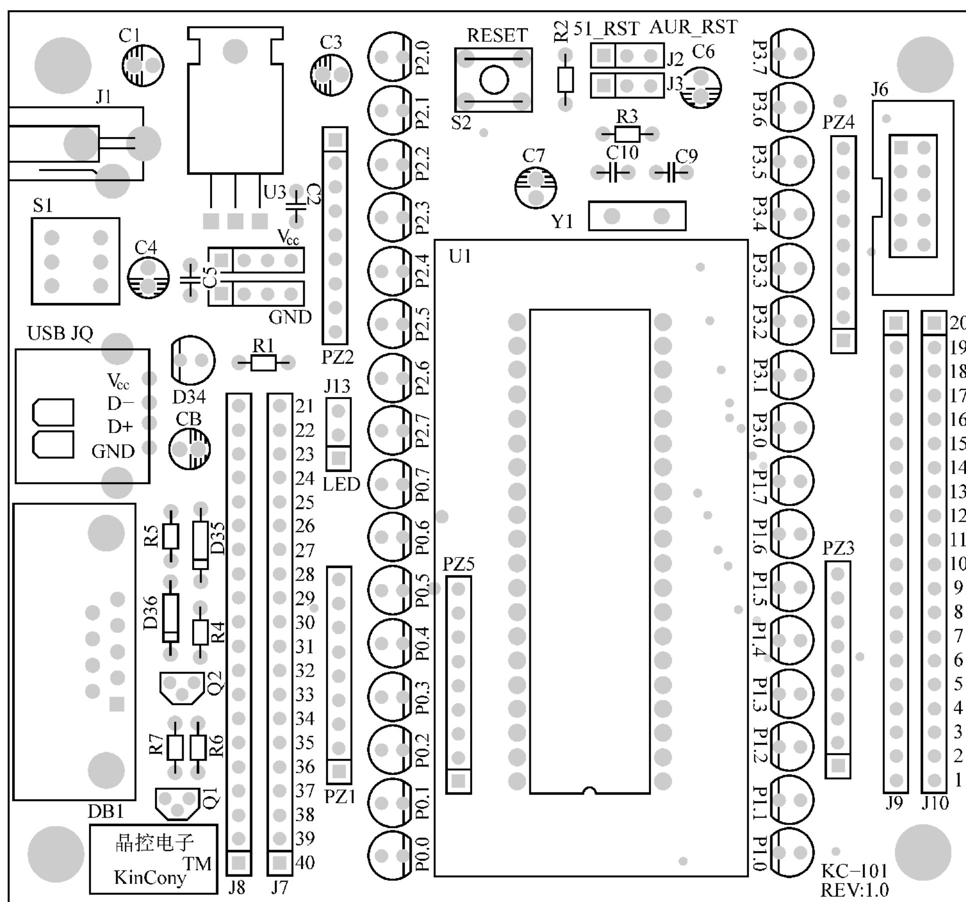
J6：ISP 下载接口：标准 10PIN ISP 下载接口，支持常为常见的 Atmel 89S 系列单片机的芯片。电路原理与元器件分布如图 11-2 所示。



a)

图 11-2　电路原理和元器件分布

a）电路原理



b)

图 11-2 （续）

b）元器件分布

4. 模块相关背景知识

常见的 MSC-51 单片机中一般采用双列直插（DIP）封装，共 40 个引脚。图 11-3 为引脚排列图。其中的 40 个引脚大致可以分为四类：电源、时钟、控制和 I/O 引脚。

（1）电源

1）Vcc：芯片电源，一般为 +5V。

2）Vss：接地端。

（2）时钟

XTAL1 和 XTAL2 是晶体振荡电路反相输入端和输出端。当使用内部振荡电路时，需要外接晶振，常见有的 4M、6M、11.0592M、12M 等。当使用外部振荡输入时 XTAL1 接地，XTAL2 接外部振荡脉冲输入。

（3）控制线

MCS-51 单片机的控制线共有 4 根，其中 3 根是复用线，具有两种功能。

1）ALE/PROG：地址锁存允许/编程脉冲。

ALE：正常使用时为 ALE 功能，主要用来锁存 P0 口送出的 8 位地址。P0 口一般分时传送低 8 位地址和数据信号，且均为二进制数。区分是否是低 8 位数据信号还是地址信号，要看 ALE 引脚。当 ALE 信号有效时，P0 口传送的是低 8 位地址信号；当 ALE 无效时，P0 口传送的是 8 位数据信号。一般在 ALE 引脚的下降沿锁定 P0 口传送的内容，即低 8 位地址信号。

当 CPU 不执行访问外部 RAM 指令（MOVX）时，ALE 以时钟振荡频率 1/6 的固定频率输出，所以 ALE 信号也可以作为外部芯片的时钟信号。但当 CPU 执行访问外部 RAM（MOVX）时，ALE 将跳过一个 ALE 脉冲。

PROG：当单片机在编程期间，该引脚输入编程脉冲（由编程器提供）。

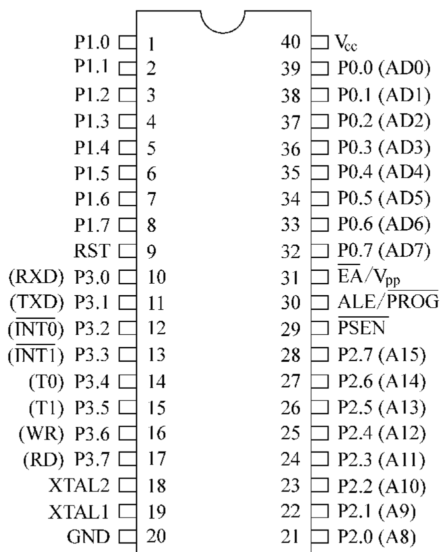


图 11-3　引脚排列图

2）PSEN：外部 ROM 读选通信号。

当单片机读外部 ROM 时，每个机器周期内 PSEN 两次有效输出。PSEN 就相当于外部 ROM 芯片输出允许的选通信号。但读片内 ROM 和读片外 RAM 时无效。

3）RST：复位引脚。

RST 为单片机上电复位输入端，只要在该引脚上连续保持两个机器周期以上的高电平，单片机就可以实现复位操作，复位后程序从 0000H 处开始执行。在一般应用中可以用 RC 电路来实现单片机的上电复位，在一些工业控制等要求较高的场合，一般用专用的看门狗芯片进行复位及电源监控，典型的 RC 上电复位电路如图 11-4 所示。

4）EA/VPP：内外 ROM 选择/EPROM 编程电源。

EA：正常工作时，EA 为内外 ROM 选择端。MCS-51 型单片机的寻址范围为 64KB，其中 4K 在片内，60K 在片外。当 EA 为高电平时，先访问片内 ROM，当程序长度超过 4K 时，将自动转向执行外部 ROM 中的程序。当 EA 为低电平时，单片机只访问外部 ROM，对老的 8031 单片机（因片内没有 ROM），EA 必须接地。目前，大部分单片机都自带 ROM，所以一般应用中也将 EA 接高电平。

VPP：对于有内部 EPROM 的单片机，在片内 EPROM 编程期间，此引脚用于施加编程电源。

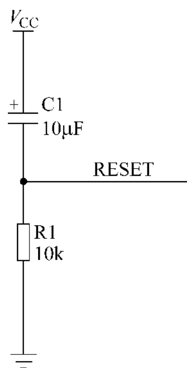


图 11-4　典型复位电路

（4）I/O 引脚

MCS-51 单片机共有 4 个 8 位并行 I/O 端口，共 32 个可编程 I/O 引脚。4 个 I/O 口各有各的功能，在一般情况下，P0 专用于分时传送低 8 位地址信号和 8 位数据信号，P2 口专用于传送高 8 位地址信号，P3 口大部分时间用于第二功能。当然所有 I/O 口都可以做为普通的输入/输出端口用。

5. 实验例程

1）例程功能：P0.0 口的 LED 发光管闪动实验。

```
01 #include <reg51.h>
02
03 sbit LED = P0^0;
04
05 void Delay()
06 {
07       unsigned char i,j;
08          for(i=0;i<255;i++)
09             for(j=0;j<255;j++);
10 }
11
12 void main()
13 {
14       while(1)
15       {
16       LED=0;
17       Delay();
18       LED=1;
19       Delay();
20    }
21 }
```

2）代码分析

序号 1：包含 51 单片机寄存器定义的头文件

序号 3：位定义 LED 为 I/O 口 P0.0

序号 5～10：一个延时函数，具体延长的时间和使用的晶体相关

序号 7：定义两个无符号变量 i，j

序号 8～9：通过 i，j 的自加嵌套循环执行，达到延时目的

序号 12～21：main 函数

序号 14：进入主程序的 while 循环

序号 16：点亮 LED

序号 17：调用延时程序

序号 18：熄灭 LED

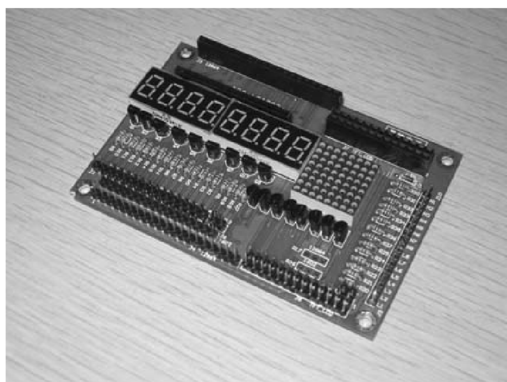
序号 19：调用延时程序

11.2　KC-102 单片机显示板模块

1. 功能描述

KC-102 为单片机显示板模块，它包括了单片机不同的显示功能组件接口，包括数码管、

液晶屏、LED 点阵屏、TFT 彩屏接口，我们用它来进行各类显示实验。KC-102 单片机显示板模块如图 11-5 所示。



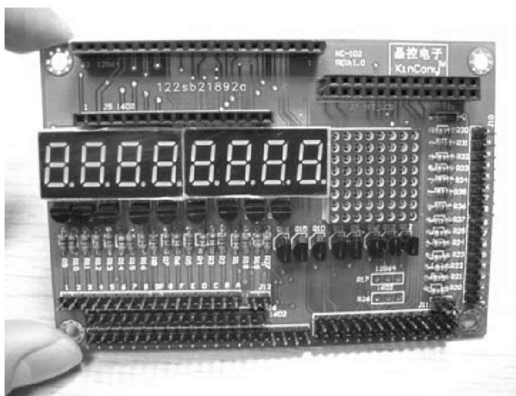


图 11-5　KC-102 单片机显示板模块

2. 硬件资源描述

1）8 位 LED 数码管：可以试验和仿真各种计数器、数字显示、以及用单片机做电子钟等仿真。比如计数器、秒表、电子钟等等。

2）1602 字符型液晶屏接口：2 行每行 16 个字符。自带字符库、带背光，经典的液晶显示器件通过液晶屏显示你想要的信息，比发光管、数码管显示更为漂亮，专业化。

3）12864 点阵型液晶接口：电子市场常见的 128X64 标准液晶模块，我们可以用来显示中文和图形。

4）点阵屏：板载 8 ×8 LED 点阵屏，可以完成点阵实验。

5）TFT 彩屏接口：1. 8in TFT 真彩液晶屏，完成单片机驱动 TFT 真彩液晶实验，制作数码相框等。

6）所有硬件资源接口，全部通过插针外延供外扩展使用。

3. 硬件接口描述

J3：12864 点阵型液晶显示屏接口，与屏直接相连。

J5：1602 字符型液晶显示屏接口，与屏直接相连。

J7：TFT 彩色液晶显示屏接口，与屏直接相连。

R26：1602 液晶显示屏对比度调节电位器。

R17：12864 液晶显示屏对比度调节电位器。

J2（1 ~8）：8 位数码管片选信号端。

J2（DP ~ A）：8 位数码管字形控制信号线。

J6：1602 液晶显示屏所有引脚。

J4：12864 液晶显示屏所有引脚。

J8：TFT 彩色液晶显示屏所有引脚。

J9：8 ×8 点阵显示屏 8 条列线。

J10：8 ×8 点阵显示屏 8 条行线。

J11、J12：VCC 与 GND 接线端。

4. 电路原理与元件分布

如图 11-6 所示。

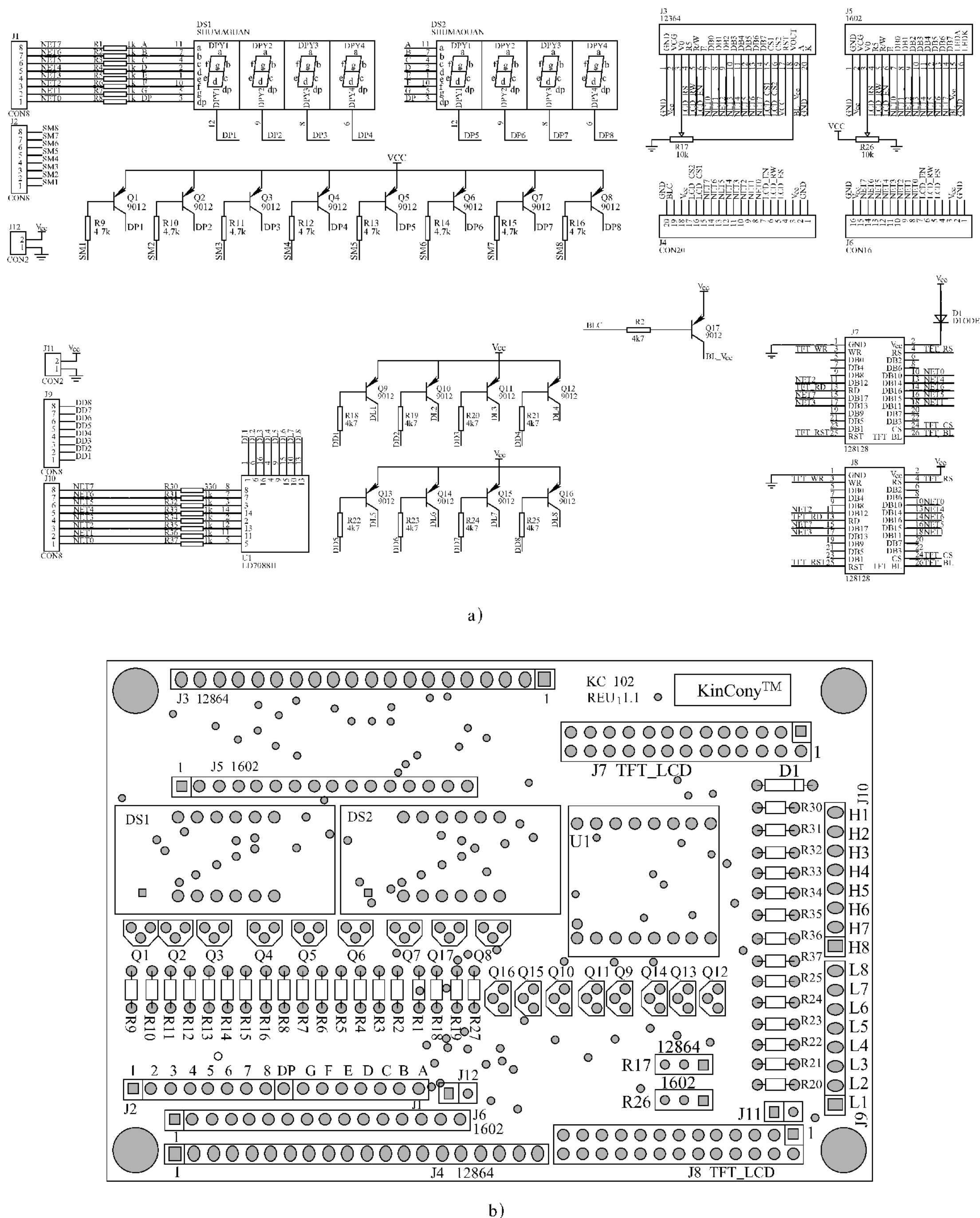


图 11-6　电路原理和元器件分布

a）电路原理　b）元器件分布

5. 模块相关背景知识

数码管、1602 字符型液晶屏、12864 点阵型液晶屏的接线及使用方法请参考书本前面部

分。

（1）8×8 点阵 LED

图 11-7 为 8×8 点阵 LED 外观及引脚，其等效电路如图 11-8 所示，8×8 点阵共需要 64 个发光二极管组成，且每个发光二极管是放置在行线和列线的交叉点上，当对应的某一列置 1 电平，某一行置 0 电平，则相应的二极管就亮，即只要其对应的 X、Y 轴顺向偏压，就可以使 LED 发亮。如果想使左上角 LED 点亮，则 Y0 = 1，X0 = 0 即可，应用时限流电阻可以放在 X 轴或 Y 轴。

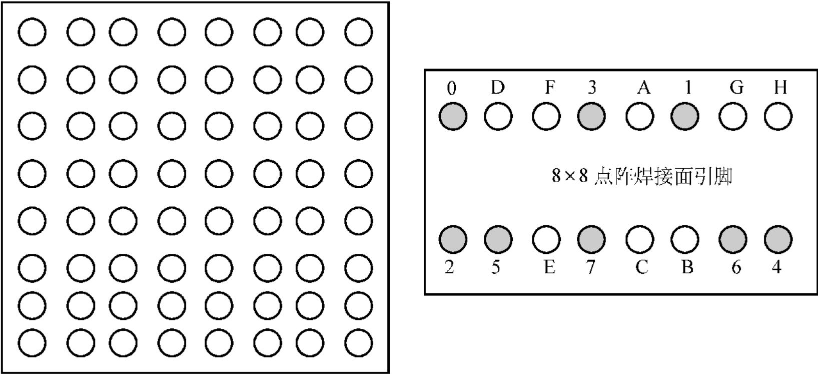


图 11-7　8×8 点阵 LED 外观及引脚

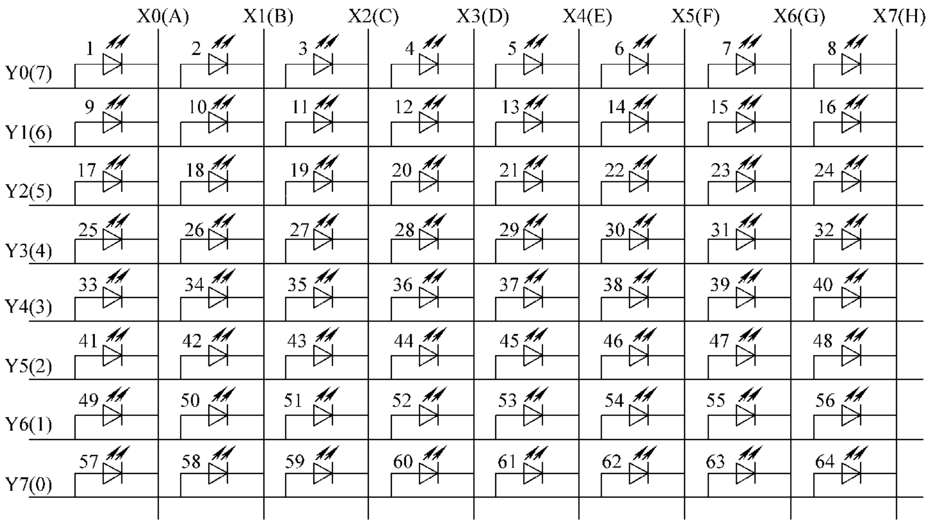


图 11-8　8×8 点阵 LED 等效电路

（2）点阵 LED 扫描法介绍

LED 一般采用扫描式显示，实际运用分为三种方式：点扫描、行扫描、列扫描。16×64 = 1024Hz，周期小于 1ms 即可。若使用第二和第三种方式，则频率必须大于 16×8 = 128Hz，周期小于 7.8ms 即可符合视觉暂留要求。此外一次驱动一列或一行（8 颗 LED）时需外加驱动电路提高电流，否则 LED 亮度会不足。

（3）设计实例

要实现一根柱形的亮法，对应的一列为一根竖柱，或者对应的一行为一根横柱，因此实现柱的亮的方法如下所述：

一根竖柱：对应的列置 1，而行则采用扫描的方法来实现。

一根横柱：对应的行置 0，而列则采用扫描的方法来实现。

硬件电路连接方法如图 11-9 所示。

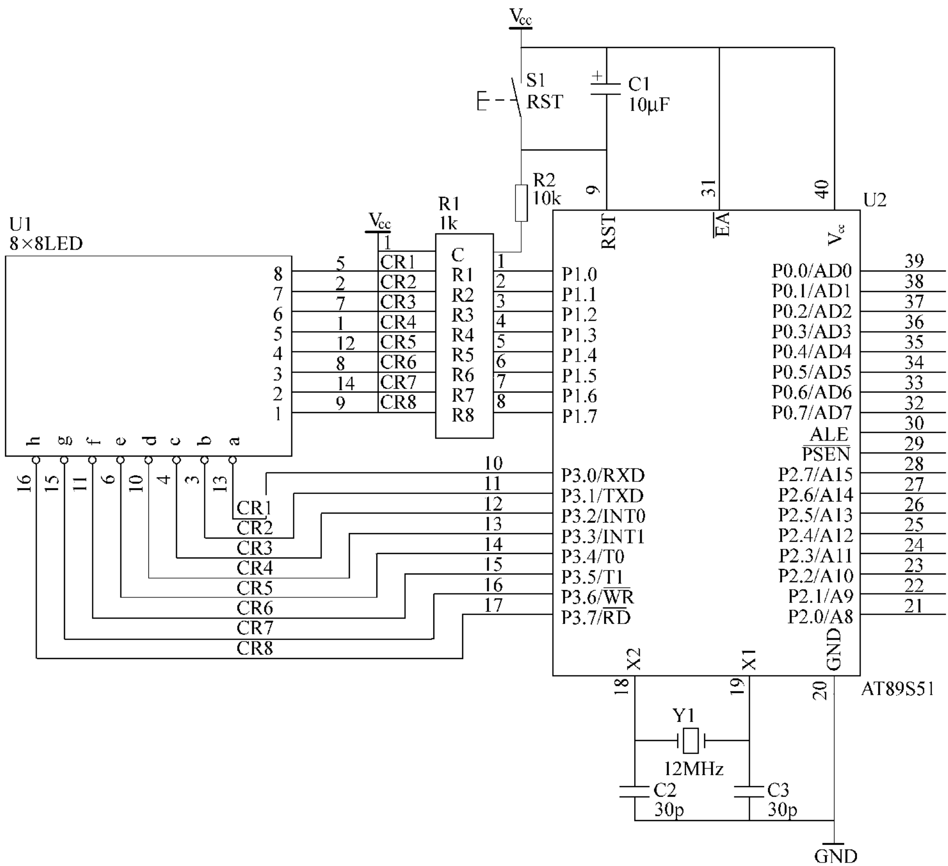


图 11-9　硬件电路连接方法

6. C 语言程序代码实例

```
#include  <AT89X52. H>

    unsigned char code taba[ ] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char code tabb[ ] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

  void delay(void)
{
  unsigned char i,j;
```

```
    for(i=10;i>0;i--)
    for(j=248;j>0;j--);
}

    void delay1(void)
{
    unsigned char i,j,k;
    for(k=10;k>0;k--)
    for(i=20;i>0;i--)
    for(j=248;j>0;j--);
}

    void main(void)
{
    unsigned char i,j;
    while(1)
    {
    for(j=0;j<3;j++)                    //从左到右3次
    {
      for(i=0;i<8;i++)
      {
        P3=taba[i];
        P1=0xff;
        delay1();
      }
    }

    for(j=0;j<3;j++)                    //从右到左3次
    {
      for(i=0;i<8;i++)
      {
        P3=taba[7-i];
        P1=0xff;
        delay1();
      }
    }
    for(j=0;j<3;j++)                    //从上到下3次
    {
      for(i=0;i<8;i++)
```

```
    {
      P3 = 0x00;
      P1 = tabb[7 - i];
      delay1();
    }
  }
  for(j = 0;j < 3;j + +)                    //从下到上 3 次
  {
    for(i = 0;i < 8;i + +)
      {
      P3 = 0x00;
      P1 = tabb[i];
      delay1();
      }
    }
  }
}
```

7. TFT 彩色显示屏

TFT（Thin Film Transistor）即薄膜场效应晶体管。所谓薄膜晶体管，是指液晶显示器上的每一液晶像素点都是由集成在其后的薄膜晶体管来驱动。从而可以做到高速度、高亮度、高对比度显示屏幕信息。TFT 属于有源矩阵液晶显示器。

TFT-LCD 液晶显示屏是薄膜晶体管型液晶显示屏，也就是“真彩”（TFT）。TFT 液晶为每个像素都设有一个半导体开关，每个像素都可以通过点脉冲直接控制，因而每个节点都相对独立，并可以连续控制，不仅提高了显示屏的反应速度，同时可以精确地控制显示色阶，所以 TFT 液晶的色彩更真。TFT 液晶显示屏的特点是亮度好、对比度高、层次感强、颜色鲜艳，但也存在着比较耗电和成本过高的不足。TFT 液晶技术加快了手机彩屏的发展。新一代的彩屏手机中很多都支持 65536 色显示，有的甚至支持 16 万色显示，这时 TFT 的高对比度，色彩丰富的优势就非常重要了。

TFT 屏的那么多点都是由硅片控制的，它就是 LCD 的控制芯片。例如要想显示左上角第一个点，那么这个点的坐标就是 X0，Y0。微处理器首先通过数据接口发送 X 坐标给 LCD，再发送 Y 给 LCD，最后发送数据组成颜色代码给 LCD。这样 LCD 就会在左上角第一个点显示出来相应的颜色。无论是一幅图片还是一个视频都是按照这样的方式来显示的。

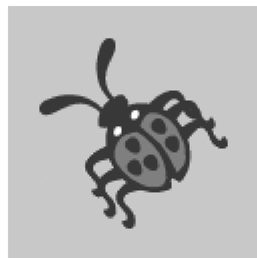


图 11-10　TFT 彩屏显示彩图

我们提供了 C 语言程序代码实例，其功能是用 51 单片机控制 TFT 彩屏显示彩图，要显示的图形如图 11-10 所示。

我们将“KC-101 51/AVR 单片机最小系统核心板”与“KC-102 单片机显示板模块”相连，用 51 单片机控制 TFT 彩屏进行图像显示，图 11-11 ~ 图 11-24 为实验全过程演示图。

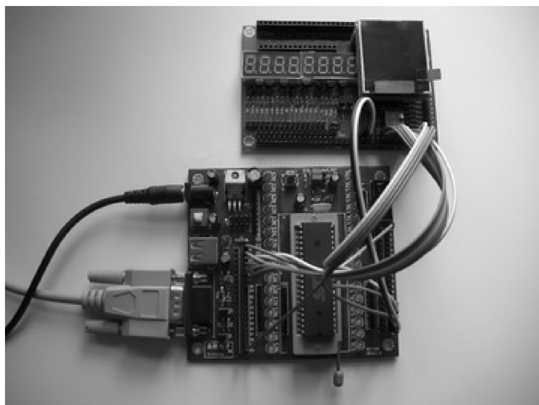


图 11-11　将 KC-101 最小系统板与 KC-102 显示模块用杜邦线进行连接

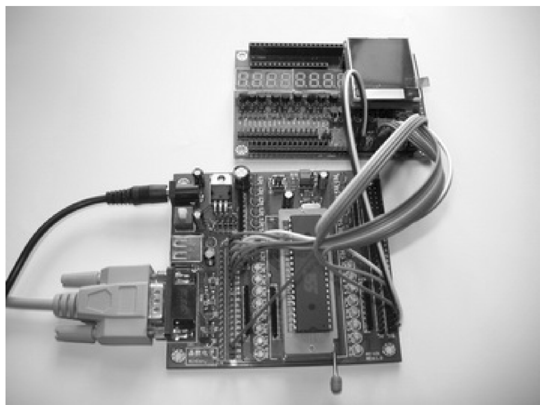


图 11-12　插上我们提供的 TFT 彩屏

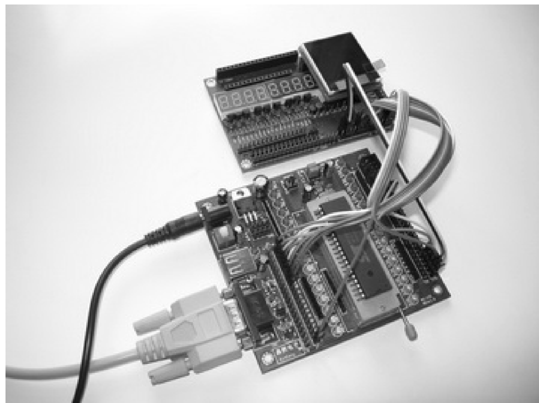


图 11-13　插上单片机芯片、外接电源和串口线

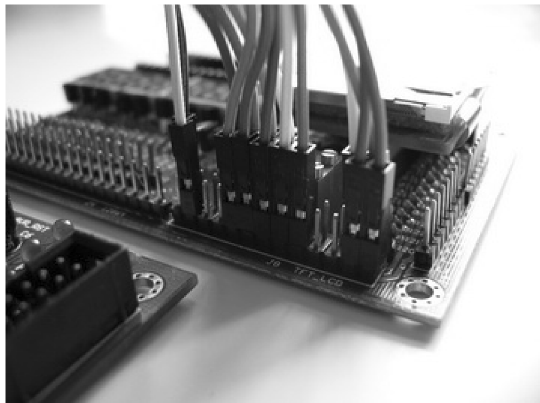


图 11-14　用杜邦线将显示模块与 KC-101 最小系统板相连

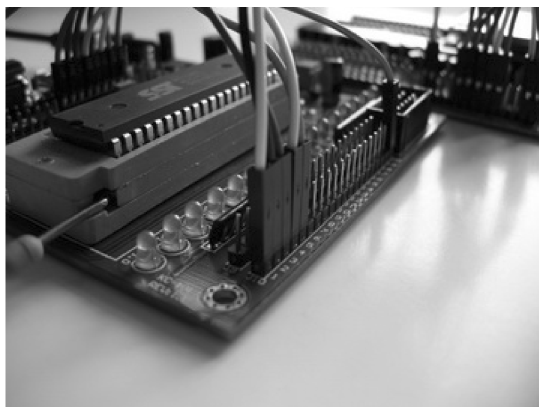


图 11-15　部分连接图片外观-1

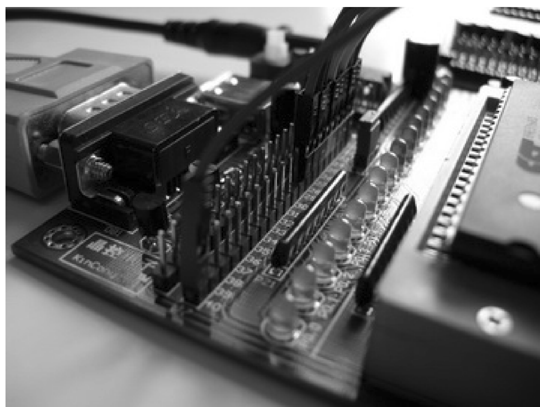


图 11-16　部分连接图片外观-2

打开电源开关，KC-101 板上红色指示灯亮，TFT 彩屏显示初始白屏状态，如图 11-17 所示。

程序运行后，TFT 彩屏出现我们要显示的图片，如图 11-18 所示。

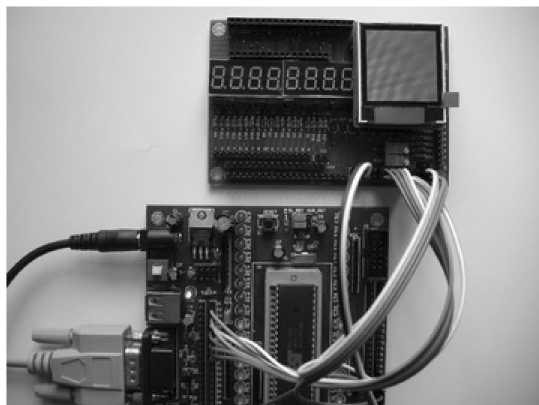


图 11-17　程序运行前状态

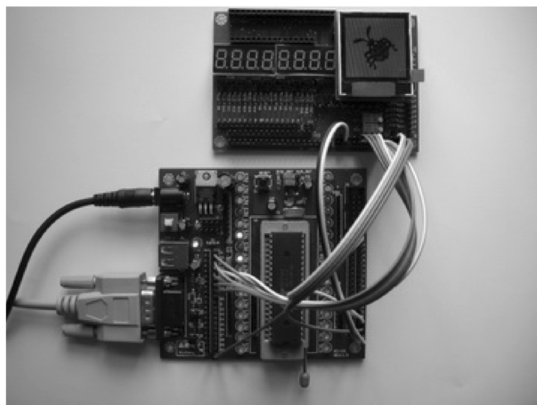


图 11-18　程序运行后彩屏显示图形

用肉眼观察 TFT 彩屏效果很漂亮，就像我们看到手机屏幕一样，如图 11-19 所示。

图片上看到的水波纹用肉眼根本看不出来，这是因为数码相机分辨率因素，如图 11-20 所示。

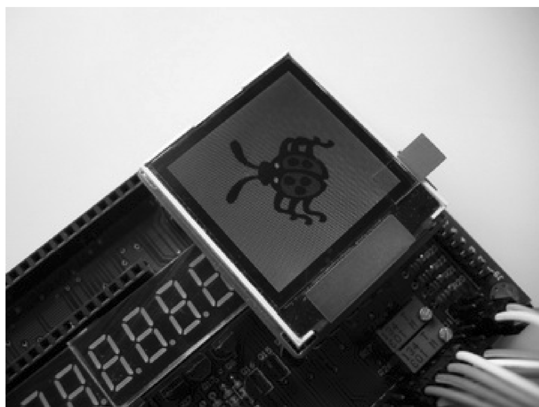


图 11-19　图像显示 1

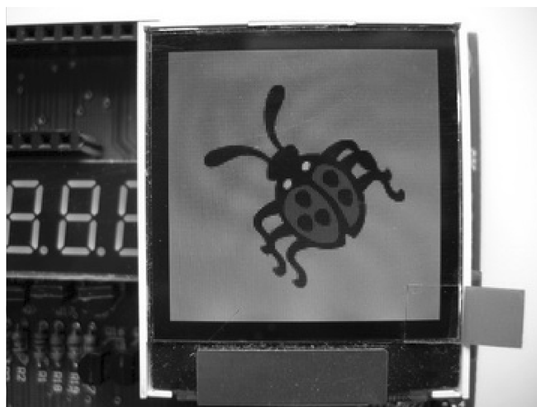


图 11-20　图像显示 2

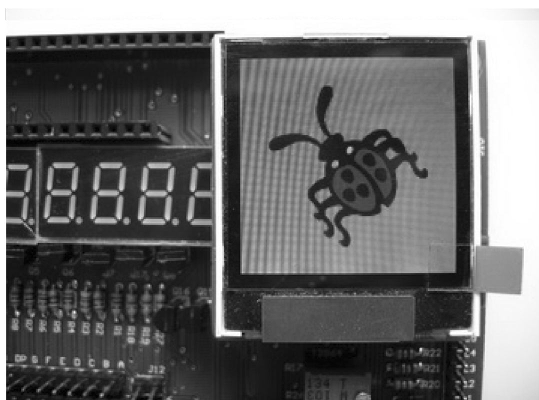


图 11-21　实际演示的瞬间拍摄照片-1

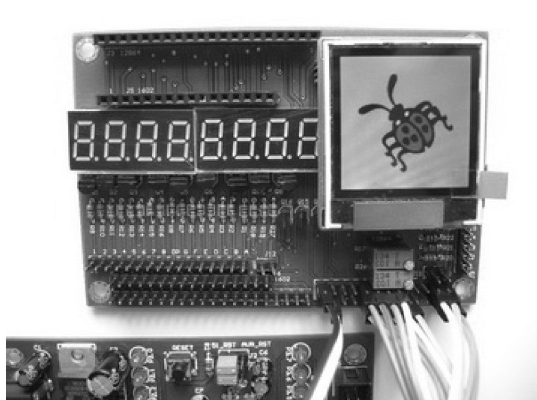


图 11-22　实际演示的瞬间拍摄照片-2

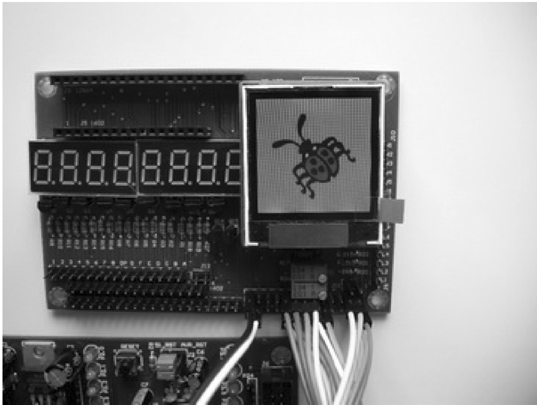


图 11-23　实际演示的瞬间拍摄照片-2

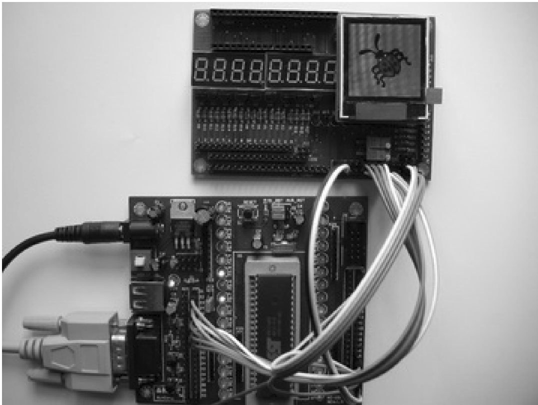
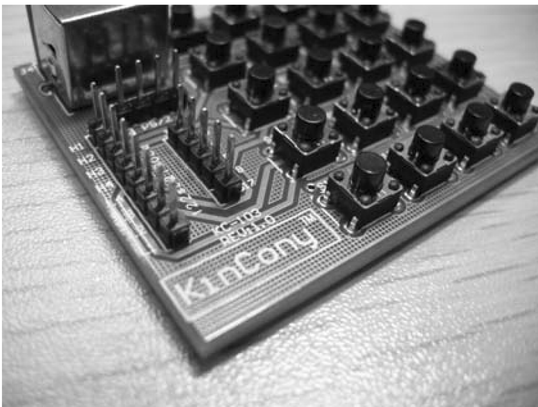


图 11-24　实际演示的瞬间拍摄照片-4

11.3　KC-103 单片机键盘板模块

1. 功能描述

KC-103 为单片机按键板，可以与 KC-101 51 单片机最小系统核心板进行连接，板子小巧，做工优良，板子按键种类齐全，包括直控按键、矩阵键盘、PS/2 键盘接口，同时带有外扩展插针。KC-103 单片机键盘板模块如图 11-25 所示。



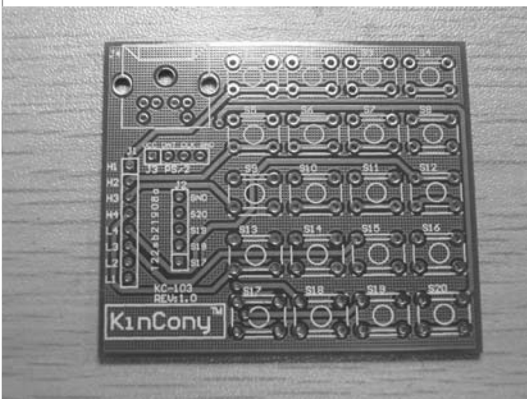


图 11-25　KC-103 单片机键盘板模块

2. 硬件资源描述

1）4 个 I/O 口直控按键：非常实用的键盘，通过简洁的程序即可完成键盘输入控制，编程方面更不需要像矩阵键盘那样绞尽脑汁。

2）4×4 矩阵键盘接口：共 16 个键位，可以试验和仿真相关教程的键盘有关的程序。

3）PS/2 链盘接口：支持 PS/2 接口的 104 键标准键盘的解码试验，大家也许还觉得上述

这些不够过瘾，那就再看看板上的 PS2 接口，一个足以让常用按钮键盘退休的强悍接口。通过随机光盘中的例程，大家一旦玩转了 PS 接口键盘，我们会突然发现单片机竟然可以接这么强大的键盘。写好程序就一切“OK”了，此例程是一个“键盘指法练习器”。在此，我们会发现单片机简直是个万能器件，它真是太神奇了。

3. 硬件接口描述

J1（H1 ~ H4）：矩阵键盘的 4 条行线。

J1（L1 ~ L4）：矩阵键盘的 4 条列线。

J2：直控按键（S17 ~ S20）的接口。

J3：PS/2 标准键盘接口。

4. 电路原理与元器件分布

电路原理和元器件分布如图 11-26 所示。

a)

图 11-26　电路原理和元器件分布

a）电路原理

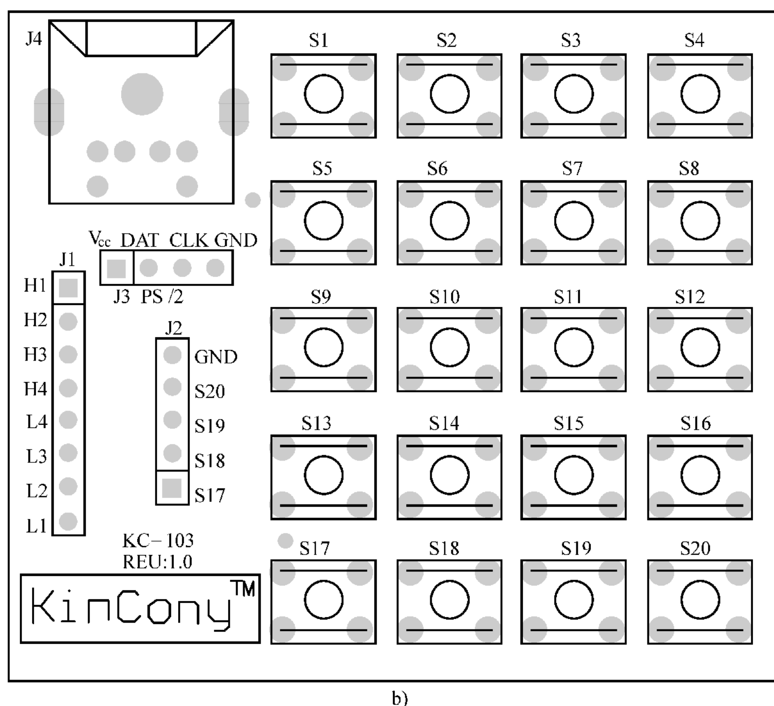


图 11-26 （续）

b）元器件分布

5. 模块相关背景知识

直控按键、矩阵键盘的接线及使用方法请参考本书前面部分。

（1）PS/2 键盘接口

早期的键盘实际上是一种 5pin 的键盘，称为 AT 键盘，是 1984 年 IBM PC 的标准键盘。在 1987 年 IBM 进行了改进，从而变成了现在的 PS/2 键盘。

当按下一个键或释放一个键，键盘都会发送键盘扫描码到主机。比如按 A，键盘就会发送 0x1C 到主机。如果持续按 A，当经过一个给定时间后，就会发送 0x1C 到主机。当键盘被释放，键盘会发送 0xF0 加键码到主机，告诉主机键盘哪个键被释放。当再次按下 A，键盘就会再次发送 0x1C 到主机。键盘的每一个键都有一个特定的键码，无论 SHIFT、Num Lock、Caps Lock、Scroll Lock 键是否被按下，键盘总是发送同样的键码，主机的键盘 BIOS 负责区分 SHIFT、Num Lock、Caps Lock、Scroll Lock 键的状态。键盘有 101 个键，而 PS/2 接口只有 8 比特。因此，并不是所有的键都只有一个字节的键码。扩展键盘中有一些键的键码是双字节的，以 E0 开头，比如向左键为 E06B。有些键的扫描码非常夸张，比如 Pause Brk 键的键码为 E1177E1F014F077！。

（2）PS/2 协议

PS/2 协议是外设与主机之间通信的一种同步双向串行协议。在该协议中主机拥有较高的优先级，在一定条件下可以终止外设正在进行的发送过程。PS/2 协议采用的传送数据帧的格式为：1 位起始位（0）、8 位数据位、1 位奇偶校验位、1 位停止位（1）。数据发送时低位在前，高位在后。外设每收到主机发来的 1 帧数据，都要紧随该帧的停止位发送一个握手位 ACK（0）应答主机。然后，外设还要发 1 帧应答数据（0xF0），表示外设已经完整地

接收到了主机的命令；而主机则不需发送握手位，也不需要发送应答帧。

（3）C 语言实例程序

功能：1602 液晶显示 PS/2 键盘按键值

```
#include <at89x51.h>
#include "scancodes.h"

#define LCM_RW   P2_1   //定义 1602LCD 引脚
#define LCM_RS   P2_0
#define LCM_E    P2_2
#define LCM_Data   P0

#define Key_Data P3_4   //定义 PS/2 键盘引脚
#define Key_CLK   P3_3   //接 51 单片机的外部中断,触发方式为低电平

#define Busy   0x80   //用于检测 LCM 状态字中的 Busy 标识

void LCMInit(void);
void DisplayOneChar(unsigned char X, unsigned char Y, unsigned char DData);
void DisplayListChar(unsigned char X, unsigned char Y, unsigned char code *DData);
void Delay5Ms(void);
void Delay400Ms(void);
void Decode(unsigned char ScanCode);
void WriteDataLCM(unsigned char WDLCM);
void WriteCommandLCM(unsigned char WCLCM,BuysC);

unsigned char ReadDataLCM(void);
unsigned char ReadStatusLCM(void);
unsigned char code cdle_net[] = {"www.hificat.com"};
unsigned char code email[] = {"hificat@163.com"};
unsigned char code Cls[] = {"                "};
static unsigned char IntNum = 0;   //中断次数计数
static unsigned char KeyV;   //键值
static unsigned char DisNum = 0;   //显示用指针
static unsigned char Key_UP = 0,Shift = 0;//Key_UP 是键松开标识,Shift 是 Shift 键按下标识
static unsigned char BF = 0; //标识是否有字符被收到

void main(void)
{
```

```
unsigned char TempCyc;

Delay400Ms();    //启动等待,等 LCM 讲入工作状态
LCMInit();    //LCM 初始化
Delay5Ms();    //延时片刻(可不要)

DisplayListChar(0, 0, cdle_net);
DisplayListChar(0, 1, email);
ReadDataLCM();   //测试用句无意义
for (TempCyc =0; TempCyc <10; TempCyc + + )
        Delay400Ms();    //延时
DisplayListChar(0, 1, Cls);

IT1 =0;    //设外部中断 1 为低电平触发
EA =1;
EX1 =1;    //开中断

do
  {
     if (BF)
        Decode(KeyV);
     else
        EA =1;    //开中断
  }
while(1);
}

//写数据
void WriteDataLCM(unsigned char WDLCM)
{
ReadStatusLCM();    //检测忙
LCM_Data = WDLCM;
LCM_RS =1;
LCM_RW =0;
LCM_E =0;    //若晶振速度太高,可以在这后加小的延时
LCM_E =0;    //延时
LCM_E =1;
}
```

```
//写指令
void WriteCommandLCM(unsigned char WCLCM,BuysC)   //BuysC 为 0 时忽略忙检测
{
if (BuysC) ReadStatusLCM();   //根据需要检测忙
LCM_Data = WCLCM;
LCM_RS =0;
LCM_RW =0;
LCM_E =0;
LCM_E =0;
LCM_E =1;
}

//读数据
unsigned char ReadDataLCM(void)
{
LCM_RS =1;
LCM_RW =1;
LCM_E =0;
LCM_E =0;
LCM_E =1;
return(LCM_Data);
}

//读状态
unsigned char ReadStatusLCM(void)
{
LCM_Data =0xFF;
LCM_RS =0;
LCM_RW =1;
LCM_E =0;
LCM_E =0;
LCM_E =1;
while (LCM_Data & Busy);   //检测忙信号
return(LCM_Data);
}

void LCMInit(void)   //LCM 初始化
{
LCM_Data =0;
```

```
WriteCommandLCM(0x38,0);    //三次显示模式设置,不检测忙信号
Delay5Ms();
WriteCommandLCM(0x38,0);
Delay5Ms();
WriteCommandLCM(0x38,0);
Delay5Ms();

WriteCommandLCM(0x38,1);    //显示模式设置,开始要求每次检测忙信号
WriteCommandLCM(0x08,1);    //关闭显示
WriteCommandLCM(0x01,1);    //显示清屏
WriteCommandLCM(0x06,1);    //显示光标移动设置
WriteCommandLCM(0x0F,1);    //显示开及光标设置
}

//按指定位置显示一个字符
void DisplayOneChar(unsigned char X, unsigned char Y, unsigned char DData)
{
Y &= 0x1;
X &= 0xF;    //限制X不能大于15,Y不能大于1
if (Y) X |= 0x40;    //当要显示第二行时地址码+0x40;
X |= 0x80;    //算出指令码
WriteCommandLCM(X, 1);    //发命令字
WriteDataLCM(DData);    //发数据
}

//按指定位置显示一串字符
void DisplayListChar(unsigned char X, unsigned char Y, unsigned char code *DData)
{
unsigned char ListLength;

  ListLength=0;
Y &= 0x1;
X &= 0xF;    //限制X不能大于15,Y不能大于1
while (DData[ListLength]>0x19)    //若到达字串尾则退出
    {
      if (X <= 0xF) //X坐标应小于0xF
        {
          DisplayOneChar(X, Y, DData[ListLength]); //显示单个字符
          ListLength++;
```

```
            X + + ;
          }
      }
}

//5ms 延时
void Delay5Ms( void)
{
unsigned int TempCyc =5552;
while( TempCyc - - ) ;
}

//400ms 延时
void Delay400Ms( void)
{
unsigned char TempCycA =5;
unsigned int TempCycB;
while( TempCycA - - )
      {
         TempCycB =7269;
         while( TempCycB - - ) ;
      } ;
}

void Keyboard_out( void) interrupt 2
{
if ( ( IntNum  > 0) && ( IntNum  < 9) )
      {
         KeyV = KeyV  > > 1; //因键盘数据是低 > >高,结合上一句所以右移一位
         if ( Key_Data) KeyV = KeyV | 0x80; //当键盘数据线为 1 时为 1 到最高位
      }
IntNum + + ;
while ( ! Key_CLK);    //等待 PS/2CLK 拉高

if ( IntNum  > 10)
      {
         IntNum =0; //当中断 11 次后表示一帧数据收完,清变量准备下一次接收
         BF =1; //标识有字符输入完了
         EA =0; //关中断等显示完后再开中断 (注:如这里不用 BF 和关中断直接调 Decode( )
```

```
                则所 Decode 中所调用的所有函数要声明为再入函数)
    }
}

void Decode(unsigned char ScanCode) //注意:如 SHIFT + G 为 12H 34H F0H 34H F0H 12H,也
                                      就是说 shift 的通码 + G 的通码 + shift 的断码 + G 的断
                                      码
{
  unsigned char TempCyc;

  if (! Key_UP)          //当键盘松开时
    {
      switch (ScanCode)
        {
          case 0xF0:  //当收到 0xF0,Key_UP 置 1 表示断码开始
            Key_UP = 1;
          break;

          case 0x12:  //左 SHIFT
            Shift = 1;
          break;

          case 0x59:  //右 SHIFT
            Shift = 1;
          break;

          default:
              if (DisNum > 15)
              {
                DisplayListChar(0, 1, Cls);//清 LCD 第二行
                DisNum = 0;
              }
            if(! Shift)  //如果 SHIFT 没按下
              {
                for (TempCyc = 0;(UnShifted[TempCyc][0]! = ScanCode)&&(TempCyc <
                    59); TempCyc + +);  //查表显示
                if (UnShifted[TempCyc][0] = = ScanCode) DisplayOneChar(DisNum, 1,
                  UnShifted[TempCyc][1]);
                DisNum + +;
```

```
            }
          else   //按下 SHIFT
            {
              for（TempCyc =0;（Shifted[TempCyc][0]！ = ScanCode)&&(TempCyc <
                 59)；TempCyc + +）；   //查表显示
              if（Shifted[TempCyc][0] = = ScanCode）DisplayOneChar(DisNum，1，
                 Shifted[TempCyc][1])；
            DisNum + +；
            }

        break；
      }
   }
else
   {
     Key_UP =0；
     switch（ScanCode）   //当键松开时不处理判码,如 G 34H F0H 34H 那么第二个 34H 不
                           会被处理
      {
        case 0x12：  //左 SHIFT
          Shift =0；
        break；

        case 0x59：  //右 SHIFT
          Shift =0；
        break；
      }
   }
BF =0；   //标识字符处理完了
}
```

11.4　KC-104 模数/数模转换模块

1. 功能描述

KC-104 单片机（AD/DA）模数/数模转换模块，它包括了常用的（AD）模数转换芯片和（DA）数模转换芯片及相关外围电路，所有信号线端口外延，方便用户扩展。KC-104 AD/DA 转换模块如图 11-27 所示。

2. 硬件资源描述

1）ADC0809——8 位的、以逐次逼近原理进行模数转换的器件。其内部有一个 8 通道多

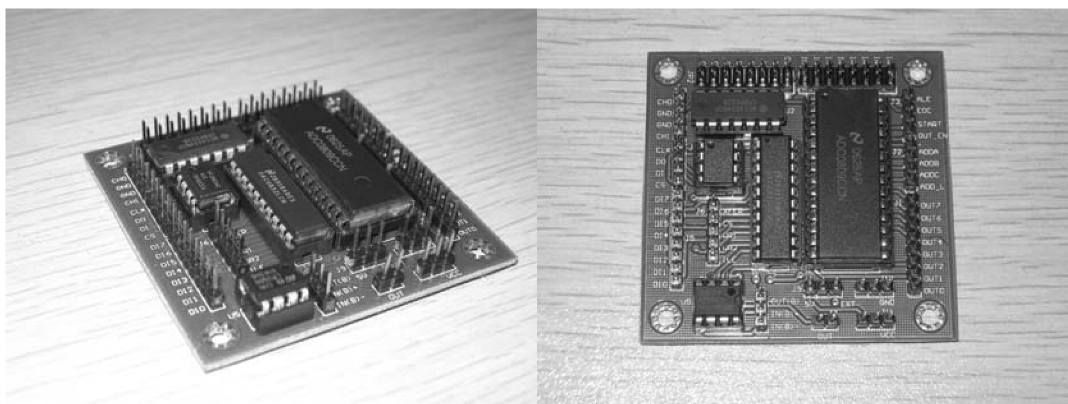


图 11-27　KC-104（AD/DA）模数/数模转换模块

路开关，它可以根据地址码锁存译码后的信号，只选通 8 路模拟输入信号中的一个进行 A/D 转换。

2）ADC0832——美国国家半导体公司生产的一种 8 位分辨率、双通道 A/D 转换芯片，其最高分辨可达 256 级，体积小、兼容性和性价比高。

3）DAC0832——8 分辨率的 D/A 转换集成芯片，与微处理器完全兼容。价格低廉、接口简单、转换控制容易等优点，在单片机应用系统中得到广泛的应用。D/A 转换器由 8 位输入锁存器、8 位 DAC 寄存器、8 位 D/A 转换电路及转换控制电路构成。

4）CD4024——7 位二进制串行计数器为 ADC0809 提供不同等级的振荡频率，即起到分频工作。

3. 硬件接口描述

J1：ADC0809 芯片的 8 位数字信号输出端。

J2：ADC0809 芯片的 ALE、ADD_A、ADD_B、ADD_C。

J3：ALE 为 CD4024 的 1 脚；EOC 为 ADC0809 芯片的 7 脚；START 为 ADC0809 芯片的 6 脚；OUT_EN 为 ADC0809 芯片的 9 脚。

J4：ADC0832 的 SPI 总线接口，分别有 CLK、DO、DI、CS。

J5：DAC0832 的 8 位数字信号输入端。

J6：DAC0832 的部分针脚。

J7：LM358 的运放输出端，DA0832 的模拟信号输出端经过 LM358 运放进行电流放大再输出。

J8：LM358 集成了两个运算放大器，一路已经与 DAC0832 相连，另一路运放资源供用户使用，J8 为供用户使用的运放输出和输出端。

J9：LM358 运放供电选择跳线：可以选择板上 VCC 5V 为运放供电电压，也可以选择“EXT”外部电源提供给运放芯片，运放的供电电压不同，数模采样转换结果不同，量化程度也会有所不同，具体电压参数关系请看 DAC0832 厂家数据手册。

J11：模块 VCC 电源端。

J12：模块 GND 地端。

JP1：ADC0809 的 8 路模拟信号输入端（IN0 ~ IN7），双排排针，靠 PCB 板外侧为 GND，右边为信号输入端。

JP2：PCB 板外侧排针为 ADC0809 的 CLOCK 时钟输入端，分别与 Q1 ~ Q7 端用短路帽连接，以便获得不同的分频值，作为 ADC0809 的时钟信号。

4. 电路原理与元器件

电路原理和元器件分布如图 11-28 所示。

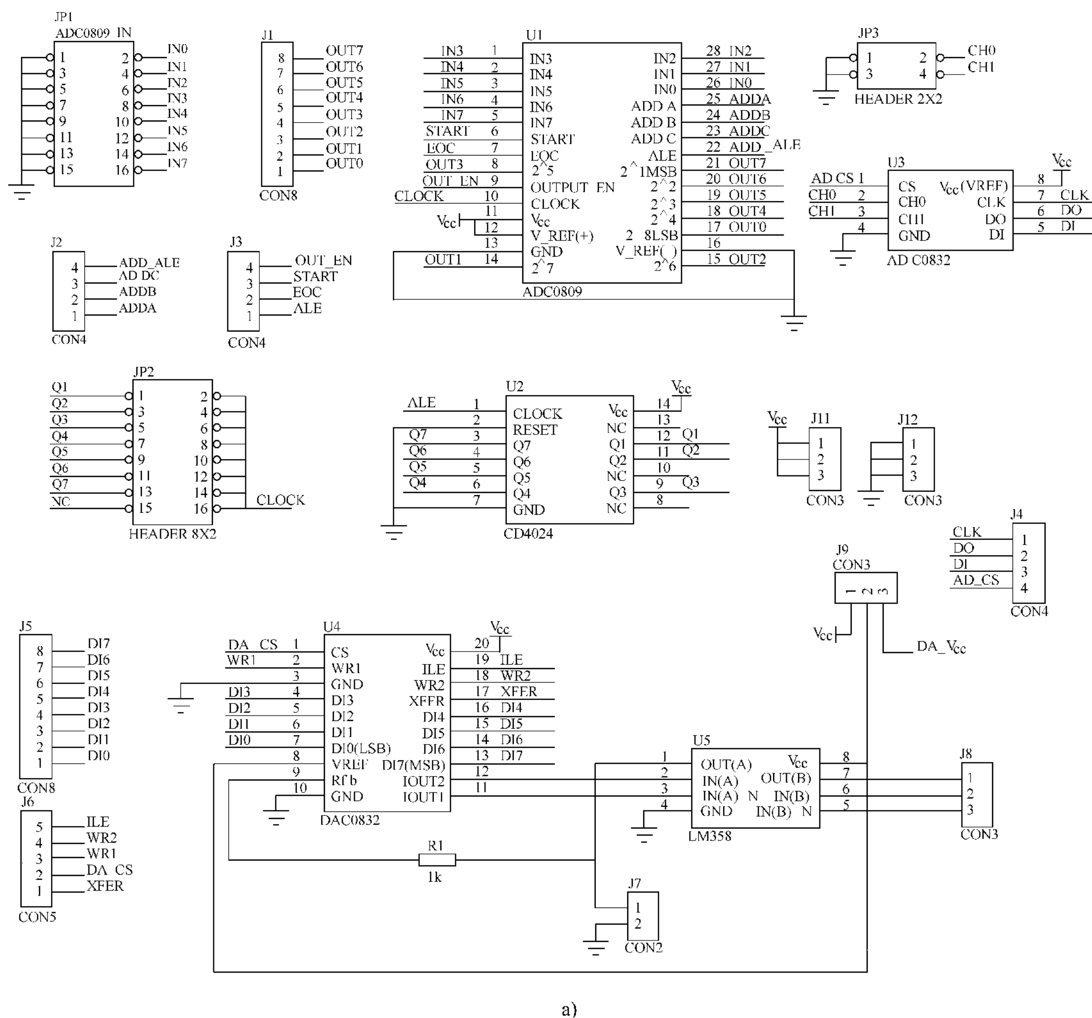
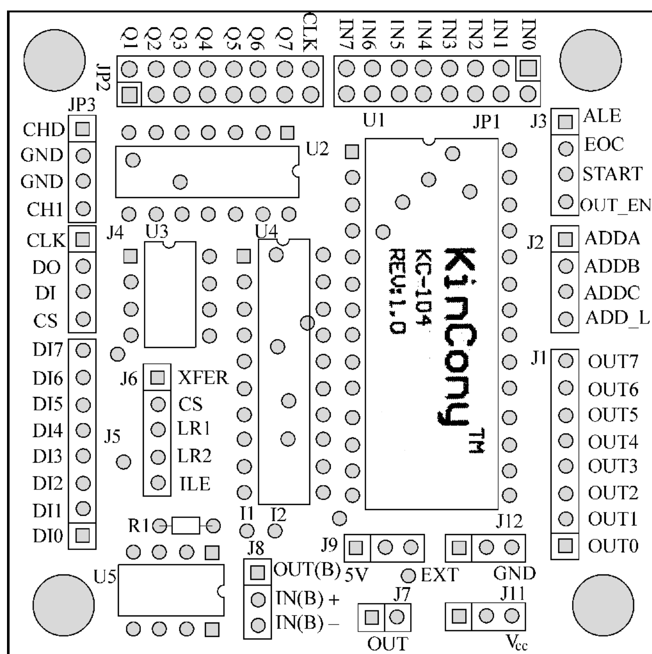


图 11-28　电路原理与元器件分布

a）电路原理



b)

图 11-28 （续）

b）元器件分布

5. 模块相关背景知识

（1）ADC0809 介绍

ADC0809 是采样分辨率为 8 位的、以逐次逼近原理进行模-数转换的器件。其内部有一个 8 通道多路开关，它可以根据地址码锁存译码后的信号，只选通 8 路模拟输入信号中的一个进行 A/D 转换。

1）主要特性

① 8 路输入通道，8 位 A/D 转换器，即分辨率为 8 位。

② 具有转换启停控制端。

③ 转换时间为 100μs。

④ 单个 +5V 电源供电。

⑤ 模拟输入电压范围为 0 ~ +5V，不需要零点和满刻度校准。

⑥ 工作温度范围为 −40 ~ +85℃。

⑦ 低功耗，约为 15mW。

2）内部结构

ADC0809 是 CMOS 单片型逐次逼近式 A/D 转换器，内部结构如图 11-29 所示，它由 8 路模拟开关、地址锁存与译码器、比较器、8 位开关树型 D/A 转换器、逐次逼近。

3）外部特性（引脚功能）

ADC0809 芯片有 28 条引脚，采用双列直插式封装，如图 11-30 所示。下面说明各引脚功能。

① IN0 ~ IN7：8 路模拟量输入端。

② 2-1 ~2-8：8 位数字量输出端。

③ ADDA、ADDB、ADDC：3 位地址输入线，用于选通 8 路模拟输入中的一路。

④ ALE：地址锁存允许信号，输入，高电平有效。

⑤ START：A/D 转换启动脉冲输入端，输入一个正脉冲（至少 100ns 宽）使其启动（脉冲上升沿使 0809 复位，下降沿启动 A/D 转换）。

⑥ EOC：A/D 转换结束信号，输出，当 A/D 转换结束时，此端输出一个高电平（转换期间一直为低电平）。

⑦ OE：数据输出允许信号，输入，高电平有效。当 A/D 转换结束时，此端输入一个高电平，才能打开输出三态门，输出数字量。

⑧ CLK：时钟脉冲输入端。要求时钟频率不高于 640kHz。

⑨ REF（+）、REF（-）：基准电压。

⑩ Vcc：电源，单一 +5V。

⑪ GND：地。

ADC0809 的工作过程是：首先输入 3 位地址，并使 ALE =1，将地址存入地址锁存器中。此地址经译码选通 8 路模拟输入之一到比较器。START 上升沿将逐次逼近寄存器复位。下降沿启动 A/D 转换，之后 EOC 输出信号变低，指示转换正在进行。直到 A/D 转换完成，EOC 变为高电平，指示 A/D 转换结束，结果数据已存入锁存器，这个信号可用作中断申请。当 OE 输入高电平时，输出三态门打开，转换结果的数字量输出到数据总线上。

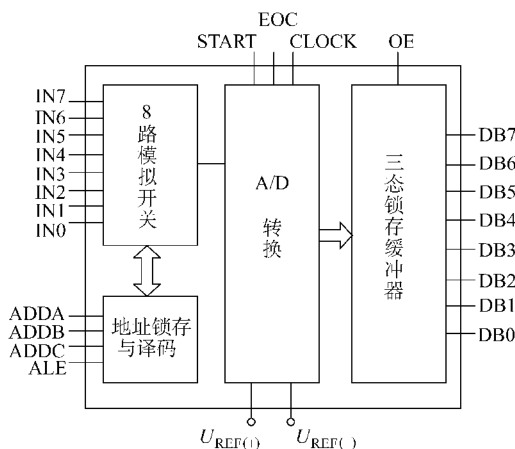


图 11-29 ADC0809 内部结构

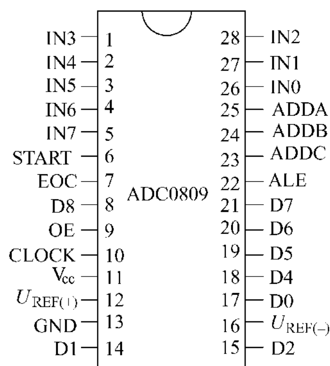


图 11-30 ADC0809 芯片

（2）DAC0832 介绍

DAC0832 是 8 分辨率的 D/A 转换集成芯片。与微处理器完全兼容。这个 DA 芯片以其价格低廉、接口简单、转换控制容易等优点，在单片机应用系统中得到广泛的应用。D/A 转换器由 8 位输入锁存器、8 位 DAC 寄存器、8 位 D/A 转换电路及转换控制电路构成。

1）DAC0832 的主要特性：

① 分辨率为 8 位。

② 电流稳定时间 1μs。

③ 可单缓冲、双缓冲或直接数字输入。

④ 只需在满量程下调整其线性度。

⑤ 单一电源供电（+5～+15V）。

⑥ 低功耗，200mW。

2）DAC0832 结构

① D0～D7：8 位数据输入线，TTL 电平，有效时间应大于 90ns（否则锁存器的数据会出错）。

② ILE：数据锁存允许控制信号输入线，高电平有效。

③ CS：片选信号输入线（选通数据锁存器），低电平有效。

④ WR1：数据锁存器写选通输入线，负脉冲（脉宽应大于 500ns）有效。由 ILE、CS、WR1 的逻辑组合产生 LE1，当 LE1 为高电平时，数据锁存器状态随输入数据线变换，LE1 的负跳变时将输入数据锁存。

⑤ XFER：数据传输控制信号输入线，低电平有效，负脉冲（脉宽应大于 500ns）有效。

⑥ WR2：DAC 寄存器选通输入线，负脉冲（脉宽应大于 500ns）有效。由 WR1、XFER 的逻辑组合产生 LE2，当 LE2 为高电平时，DAC 寄存器的输出随寄存器的输入而变化，LE2 的负跳变时将数据锁存器的内容打入 DAC 寄存器并开始 D/A 转换。

⑦ IOUT1：电流输出端 1，其值随 DAC 寄存器的内容线性变化。

⑧ IOUT2：电流输出端 2，其值与 IOUT1 值之和为一常数。

⑨ Rfb：反馈信号输入线，改变 Rfb 端外接电阻值可调整转换满量程精度。

⑩ Vcc：电源输入端，Vcc 的范围为 +5～+15V。

⑪ VREF：基准电压输入线，VREF 的范围为 -10～+10V。

⑫ AGND：模拟信号地。

⑬ DGND：数字信号地。

（3）DAC0832 的工作方式

根据对 DAC0832 的数据锁存器和 DAC 寄存器的不同的控制方式，DAC0832 有三种工作方式：直通方式、单缓冲方式和双缓冲方式。DAC 0832 如图 11-31 所示。

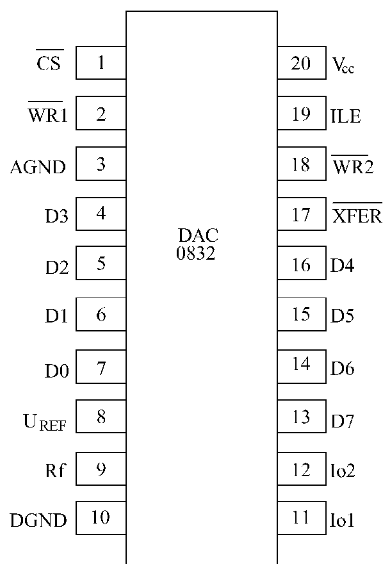


图 11-31　DAC 0832 芯片

（4）LM358 运放介绍

模块板上的 DAC0832 模数转换后，由于输出电流较小，我们在其输出端接了 LM358 运放进行放大。

LM358 内部包括有两个独立的、高增益、内部频率补偿的双运算放大器，适合于电源电压范围很宽的单电源使用，也适用于双电源工作模式，在推荐的工作条件下，电源电流与电源电压无关。它的使用范围包括传感放大器、直流增益模块和其他所有可用单电源供电的使用运算放大器的场合。

LM358 的封装形式有塑封 8 引线双列直插式和贴片式，如图 11-32 所示。

特性（Features）：

① 内部频率补偿

② 直流电压增益高（约100dB）

③ 单位增益频带宽（约1MHz）

④ 电源电压范围宽：单电源（3～30V）双电源（±1.5～±15V）

① 低功耗电流，适合于电池供电

② 低输入偏流

③ 低输入失调电压和失调电流

④ 共模输入电压范围宽，包括接地

⑤ 差模输入电压范围宽，等于电源电压范围

⑥ 输出电压摆幅大（0至Vcc-1.5V）

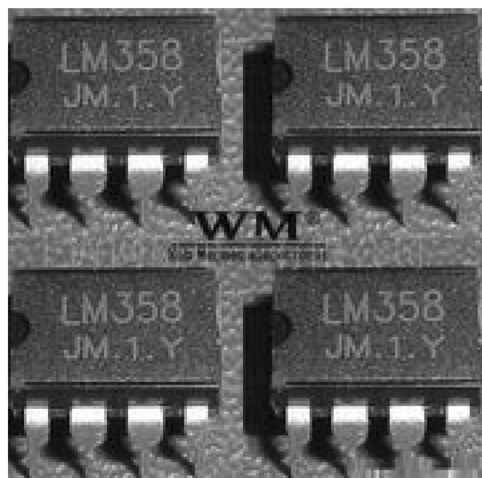


图11-32 LM358运放外观

关于ADC0832芯片介绍请看本书前面部分内容。

（5）C语言实例程序

功能：ADC0832模数转换实验，通过数码管实时显示ADC0832某一路输出端的电压值。

```
/*********************************************************************/
/*杭州晶控电子有限公司                                                 */
/*http://www.hificat.com                                              */
/*ADC0832测试程序                                                      */
/*目标器件:AT89S51                                                     */
/*晶振:11.0592MHz                                                      */
/*编译环境:Keil 7.50A                                                  */
/*********************************************************************/

/***********************包含头文件*************************/
#include <reg51.h>
#include <intrins.h>

/***********************端口定义*************************/
sbit  ADCS   = P3^4;
sbit  ADCLK  = P1^0;
sbit  ADDIO  = P1^1;


/*********************定义全局变量**********************/
```

```
unsigned int adva1;                //AD 值

/*********************共阳LED段码表*****************  /
unsigned char code tab[ ] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};



/******************************************************/
函数功能:AD 转换子程序
入口参数:CH
出口参数:adval
*********************************************  /
unsigned char Adc0832(unsigned char channel)    //AD 转换,返回结果
{
  unsigned char i;
  unsigned char dat =0;

  ADCLK =0;
  ADDIO =1;
  ADCS =0;                 //拉低 CS 端
  ADCLK =1;
  ADCLK =0;                //拉低 CLK 端,形成下降沿 1

  ADDIO = channel&0x1;
  ADCLK =1;
  ADCLK =0;                //拉低 CLK 端,形成下降沿 2

  ADDIO = (channel > >1)&0x1;
  ADCLK =1;
  ADCLK =0;                //拉低 CLK 端,形成下降沿 3


  ADDIO =1;
for(i =0;i <8;i + +)
  {
    ADCLK =1;
    ADCLK =0;              //形成一次时钟脉冲
    if(ADDIO)
    dat| = 0x80 > >i;      //收数据
```

```
    }

    ADCS = 1;                    //拉低 CS 端
    ADCLK = 1;
    ADDIO = 1;                   //拉高数据端,回到初始状态
    return(dat);                 //return dat
}

/*****************************************
函数功能:数码管显示子程序
入口参数:
出口参数:
*****************************************/
void delay1(void)
{
int k;
    for(k = 0;k < 400;k++);
}

/*****************************************
函数功能:数码管显示子程序
入口参数:k
出口参数:
*****************************************/
void display(int k)
{
P2 = 0xfe;
P0 = tab[k/1000];
delay1();
P2 = 0xfd;
P0 = tab[k%1000/100];
delay1();
P2 = 0xfb;
P0 = tab[k%100/10];
delay1();
P2 = 0xf7;
P0 = tab[k%10];
delay1();
```

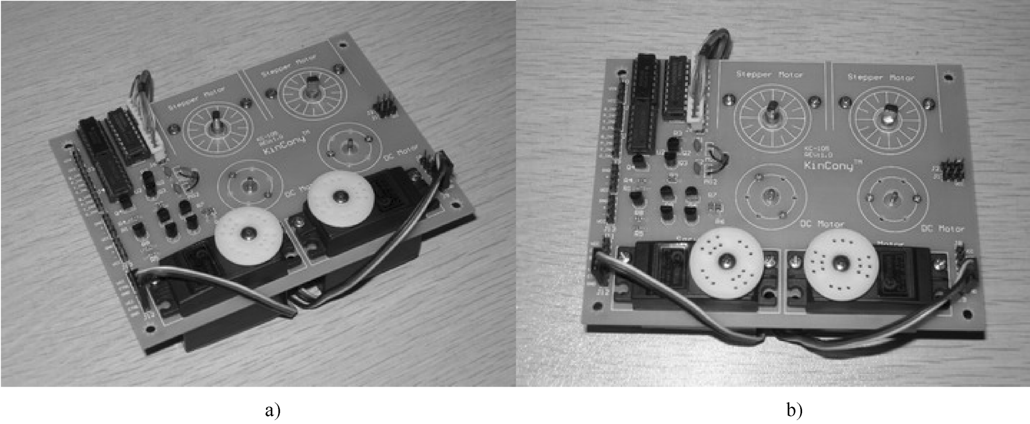
```
P2 = 0xff;
}

/****************************************
函数功能:主程序
入口参数:
出口参数:
****************************************/
void main(void)
{
  unsigned char adc;
P2 = 0xff;                    //端口初始化
P0 = 0xff;
while(1)                      //主循环
    {
       adc = Adc0832(0x01);
       display(adc);          //显示 AD 值
       }
}
```

11.5 KC-105 电动机驱动模块

1. 功能描述

KC-105 电动机驱动模块 集成了多种常用电动机，包括步进电动机，直流电动机，舵机三种类型，每种电动机均有 2 个。板子硬件结构紧凑，各自独立，硬件驱动电路设计合理，驱动接口简单明了，是开发电动机控制程序不错的选择。KC-105 电动机驱动模块如图 11-33 所示。



a)　　b)

图 11-33　KC-105 电动机驱动模块

a）正视图　b）侧视图

2. 硬件资源描述

1）步进电动机——选用 24BYJ48 微型电动机，此种电动机噪声小，温升低，重量轻，各方面性能均达到或超过国家标准。驱动方式为四相八拍一个周期，电路由 74HC04 与 ULN2803 构成，驱动能力强，无需外围电路，为提高驱动能力，我们提供了宽范围电压选择接口。在设计时，我们绘制了清晰的刻度盘，给你的学习带来更直观的感受。步进电动机刻度如图 11-34 所示。

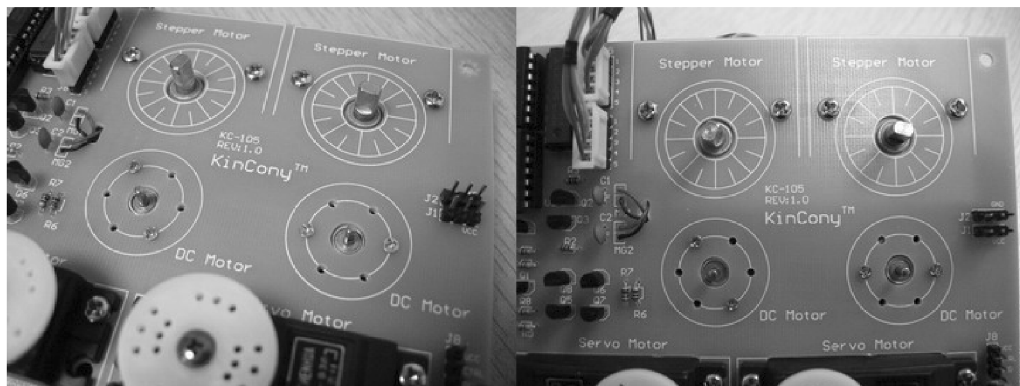


图 11-34　步进电动机、直流电动机刻度清晰可见

2）直流电动机——选用 SCRF-300 型电动机，此种电动机应用广泛，常用于一些视听设备，小家电，玩具等，如磁带录音机，电动剃须刀，电动牙刷等电路，设计了正反转控制电路，让你也玩转直流电动机。直流电动机刻度如图 11-34 所示。

3）舵机——舵机的控制信号是 PWM 信号，利用占空比的变化改变舵机的位置，引脚为三端 VCC，CTRL，GND，使用单片机可直接驱动。可配一字型、圆型转盘，标准脉冲输入。

各种类型电动机接口如图 11-35 所示。电动机安装背面如图 11-36 所示。

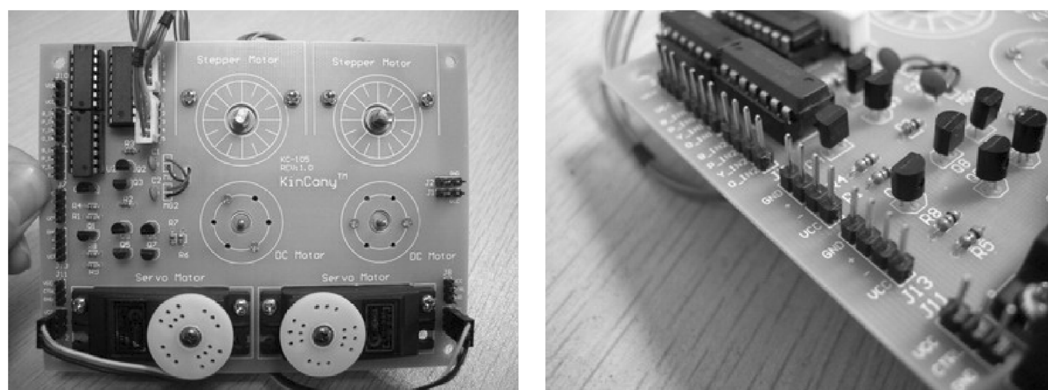


图 11-35　各种类型的电动机，接口俱全

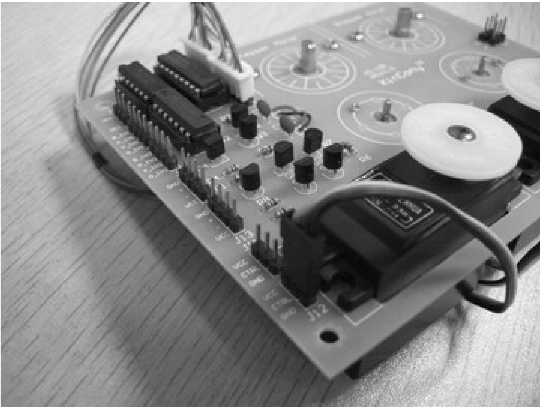
3. 硬件接口描述

J1：模块板 VCC 供电端。

J2：模块板 GND 接地端。

J3：1 号步进电动机控制端 4 条相线，与单片机直接相连。

J4：2 号步进电动机控制端 4 条相线，与单片机直接相连。



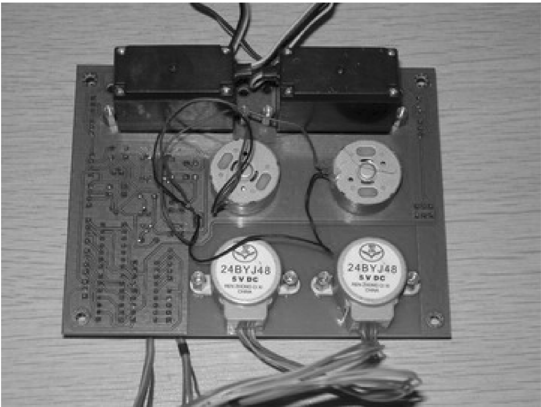


图 11-36　电动机背面安装照片

J5：1 号步进电动机接口，与步进电动机线缆相连。

J6：2 号步进电动机接口，与步进电动机线缆相连。

J7：1 号直流电动机控制接口，与单片机直接相连，用来控制直流电动机的正反转。

J13：2 号直流电动机控制接口，与单片机直接相连，用来控制直流电动机的正反转。

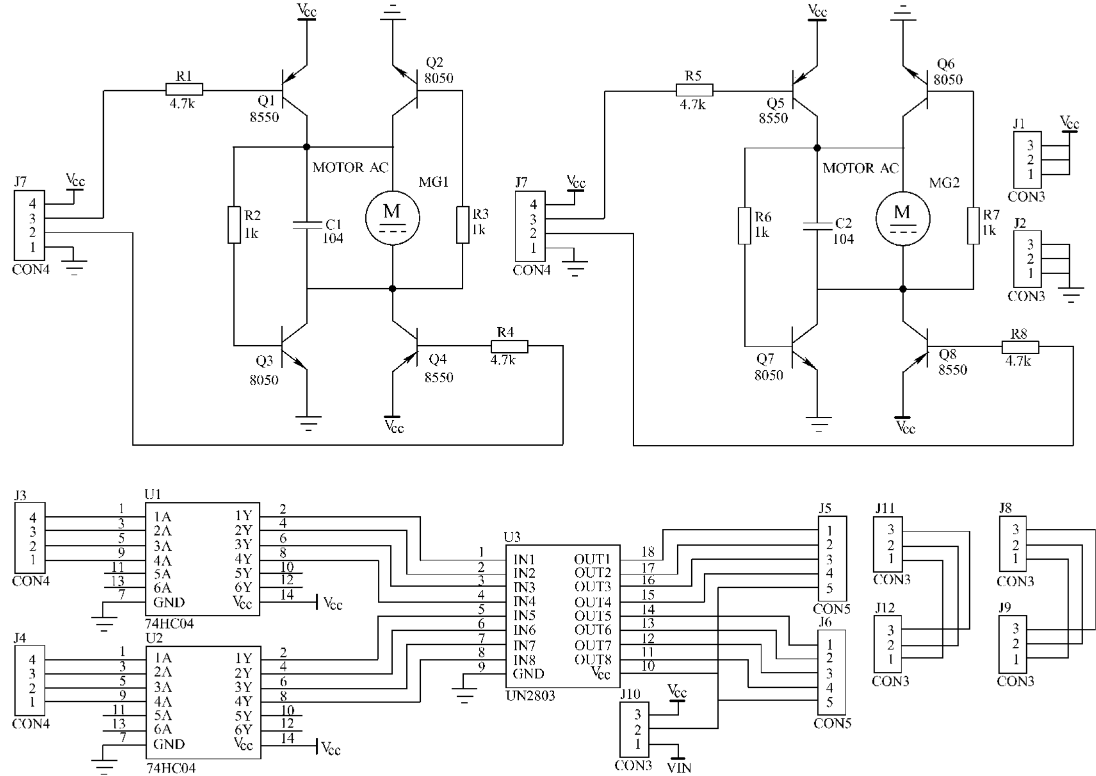
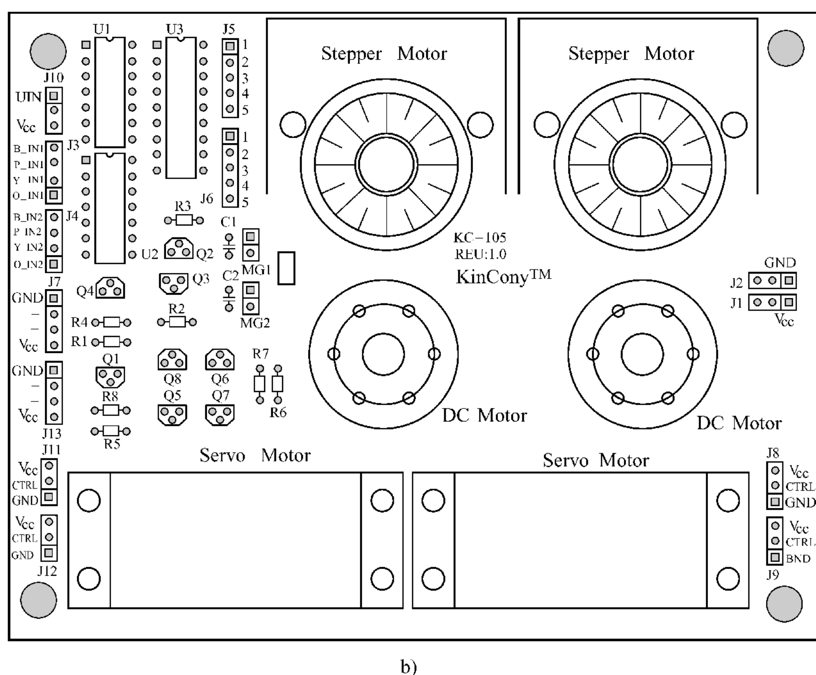


图 11-37　电路原理和元器件分布

a）电路原理



b)

图 11-37 （续）

b）元器件分布

J8：舵机信号线接口，与单片机相连。

J9：舵机信号线接口，与舵机信号线相连。

J11：舵机信号线接口，与单片机相连。

J12：舵机信号线接口，与舵机信号线相连。

4. 电路原理与元器件分布

电路原理和元器件分布如图 11-37 所示。

5. 模块相关背景知识

（1）舵机工作原理

舵机主要是由外壳、电路板、无核心电动机、齿轮与位置检测器所构成。其工作原理是由接收机发出讯号给舵机，经由电路板上的 IC 判断转动方向，再驱动无核心电动机开始转动，透过减速齿轮将动力传至摆臂，同时由位置检测器送回讯号，判断是否已经到达定位。位置检测器其实就是可变电阻，当舵机转动时电阻值也会随之改变，藉由检测电阻值便可知转动的角度。一般的伺服电动机是将细铜线缠绕在三极转子上，当电流流经线圈时便会产生磁场，与转子外围的磁铁产生排斥作用，进而产生转动的作用力。依据物理学原理，物体的转动惯量与质量成正比，因此要转动质量愈大的物体，所需的作用力也愈大。舵机为求转速快、耗电小，于是将细铜线缠绕成极薄的中空圆柱体，形成一个重量极轻的五极中空转子，并将磁铁置于圆柱体内，这就是无核心电动机。为了适合不同的工作环境，有防水及防尘设计的舵机，并且因应不同的负载需求，舵机的齿轮有塑胶及金属之区分，金属齿轮的舵机一般皆为大扭力及高速型，具有齿轮不会因负载过大而崩牙的优点。较高级的舵机会装置滚珠轴承，使得转动时能更轻快精准。滚珠轴承有一颗及二颗的区别，当然是两颗的比较好。

舵机是一种位置伺服的驱动器，适用于那些需要角度不断变化并可以保持的控制系统。工作时，控制信号由接收机的通道进入信号调制芯片，获得直流偏置电压。它内部有一个基准电路，产生周期为20ms，宽度为1.5ms的基准信号，将获得的直流偏置电压与电位器的电压比较，获得电压差输出。最后，电压差的正负输出到电动机驱动芯片决定电动机的正反转。当电动机转速一定时，通过级联减速齿轮带动电位器旋转，使得电压差为0，电动机停止转动。

控制脉冲固定周期为20ms，当送出以下正脉冲宽度时，可以得到不同的控制结果：

1）正脉冲宽度为1ms时，舵机会反转（逆时针）。

2）正脉冲宽度为2ms时，舵机会正转（顺时针）。

3）正脉冲宽度为1.5ms时，舵机会回到中点。

步进电动机的工作原理请看本书前面章节介绍。

（2）直流电动机工作原理

直流电动机是依靠直流工作电压运行的电动机，广泛应用于收录机、录像机、影碟机、电动剃须刀、电吹风、电子表和玩具等。

永磁式直流电动机也由定子磁极、转子、电刷、外壳等组成，定子磁极采用永磁体（永久磁钢），有铁氧体、铝镍钴、钕铁硼等材料。按其结构形式可分为圆筒型和瓦块型等几种。录放机中使用的电动机多数为圆筒型磁体，而电动工具及汽车用电器中使用的电动机多数采用专块型磁体。

转子一般采用硅钢片叠压而成，较电磁式直流电动机转子的槽数少。录放机中使用的小功率电动机多数为3槽，较高档的为5槽或7槽。漆包线绕在转子铁心的两槽之间（三槽即有3个绕组），其各接头分别焊在换向器的金属片上。电刷是连接电源与转子绕组的导电部件，具备导电与耐磨两种性能。永磁电动机的电刷使用单性金属片或金属石墨电刷、电化石墨电刷。

录放机中使用的永磁式直流电动机，采用电子稳速电路或离心式稳速装置。

J7和J13为直流电动机与单片机的接口，VCC和GND为电源端，中间两个引脚我们将其置为高电平“1”和低电平“0”来控制直流电动机转动，如果要换一下转动方向，那我们只要分别将其置为低电平“0”和高电平“1”即可。

11.6 KC-106单片机总线模块

1. 功能描述

KC-106单片机总线模块　包括RS232、RS485、IIC、SPI和CAN 5种常用总线接口，开放各功能引脚，模块化设计，为初学者提供一个简单明了、易学易用的总线学习平台。如图11-38所示。

2. 硬件资源描述

1）CAN总线：由Microchip公司的CAN控制芯片MCP2515和接口芯片MCP2551组合构成，MCP2515通过时钟SLK，片选CS，数据收发DI，DO 4线与MCU通信，经由MCP2551接口芯片输出，实现数据收发。

2）SPI总线：使用简单的93C46存储芯片实现SPI的应用。

3）IIC总线：使用24C02存储芯片实现IIC功能。

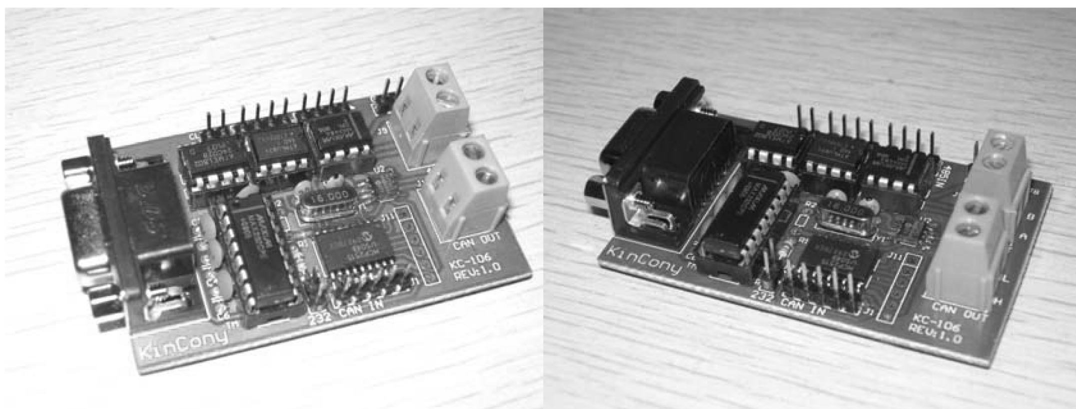


图 11-38　KC-106 单片机总线模块

4）RS232 总线：采用最常见的 MAX232 实现，外围简单，使用方便。

5）RS485 总线：使用 MAX485 芯片，也采用 4 线制的方法与 MCU 相连，无需外围电路，简单高效。

3. 硬件接口描述

J1：CAN 总线各控制数据功能引脚。

J3：串口输出端，RXD，TXD。

J4：串口输入端。

J5：SPI 4 线输入端。

J6：IIC 输入端。

J7：MAX485 输入端。

J8：两路 VCC 和 GND。

J9：MAX485 输出端。

J10：CAN 总线输出端。

J11：CAN 总线功能扩展引脚。

4. 电路原理与元器件分布

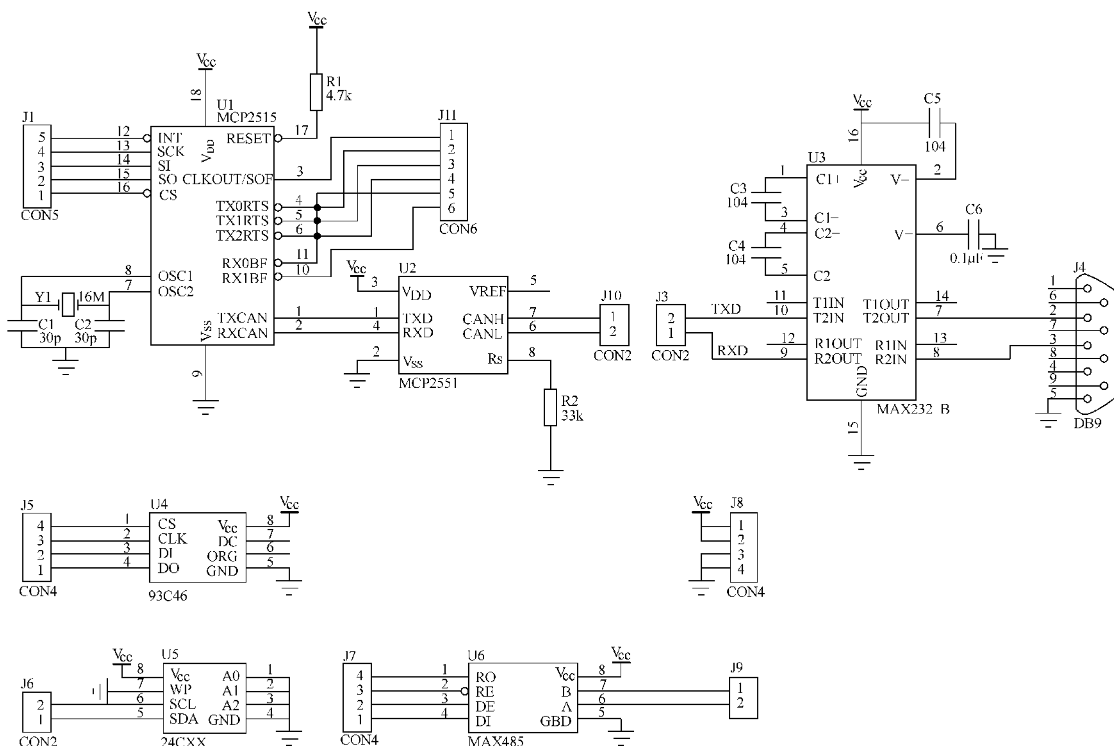
电路原理与元器件分布如图 11-39 所示。

5. 模块相关背景知识

IIC 总线、SPI 总线、RS232 串口通信相关知识详见本书前面部分内容，在这节将重点介绍 CAN 总线和 RS485 总线接口。

CAN 是 Controller Area Network 的缩写（以下称为 CAN），是 ISO 国际标准化的串行通信协议。在当前的汽车产业中，出于对安全性、舒适性、方便性、低公害、低成本的要求，各种各样的电子控制系统被开发出来。由于这些系统之间通信所用的数据类型及对可靠性的要求不尽相同，由多条总线构成的情况很多，线束的数量也随之增加。为适应“减少线束的数量”、“通过多个 LAN，进行大量数据的高速通信”的需要，1986 年德国电气商博世公司开发出面向汽车的 CAN 通信协议。此后，CAN 通过 ISO11898 及 ISO11519 进行了标准化，现在欧洲已是汽车网络的标准协议。

目前，CAN 总线的高性能和可靠性已被认同，并被广泛地应用于工业自动化、船舶、



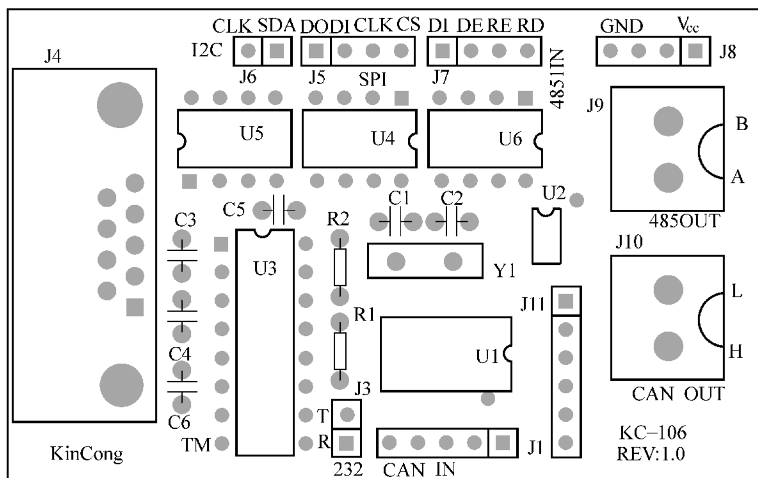


图 11-39　电路原理和元器件分布

a）电路原理　b）元器件分布

医疗设备、工业设备等方面。现场总线是当今自动化领域技术发展的热点之一，被誉为自动化领域的计算机局域网，它的出现为分布式控制系统实现各节点之间实时、可靠的数据通信提供了强有力的技术支持。

CAN 属于现场总线的范畴，它是一种有效支持分布式控制或实时控制的串行通信网络。

较之目前许多 RS485 基于 R 线构建的分布式控制系统而言，基于 CAN 总线的分布式控制系统在以下方面具有明显的优越性：

首先，CAN 控制器工作于多主方式，网络中的各节点都可以根据总线访问优先权（取决于报文标识符）采用无损结构的逐位仲裁的方式竞争向总线发送数据，且 CAN 总线协议废除了站地址编码，而代之以对通信数据进行编码，这使不同的节点同时接收到相同的数据，这些特点使得 CAN 总线构成的网络各节点之间的数据通信实时性强，并且容易构成冗余结构，提高系统的可靠性和系统的灵活性。而利用 RS485 只能构成主从式结构系统，通信方式也只能以主站轮询的方式进行，系统的实时性、可靠性较差。

其次，CAN 总线通过 CAN 收发器接口芯片 82C250 的两个输出端 CANH 和 CANL 与物理总线相连，而 CANH 端的状态只能是高电平或悬浮状态，CANL 端只能是低电平或悬浮状态。这就保证不会出现在 RS485 网络中，当系统有错误，出现多节点同时向总线发送数据时，导致总线呈现短路，从而损坏某些节点的现象。而且 CAN 总线节点在错误严重的情况下具有自动关闭输出功能，以使总线上其他节点的操作不受影响，从而保证不会出现在网络中，因个别节点出现问题，使得总线处于“死锁”状态。而且，CAN 总线具有的完善的通信协议可由 CAN 控制器芯片及其接口芯片来实现，从而大大降低了系统开发难度，缩短了开发周期，这些是只仅仅有电气协议的 RS485 所无法比拟的。

另外，与其他现场总线比较而言，CAN 总线是具有通信速率高、容易实现、且性价比高等诸多特点的一种已形成国际标准的现场总线。这些也是目前 CAN 总线应用于众多领域，具有强劲的市场竞争力的重要原因。

CAN 总线即控制器局域网络，属于工业现场总线的范畴。与一般的通信总线相比，CAN 总线的数据通信具有突出的可靠性、实时性和灵活性。由于其良好的性能及独特的设计，CAN 总线越来越受到人们的重视。它在汽车领域上的应用是最广泛的，世界上一些著名的汽车制造厂商，如 BENZ（奔驰）、BMW（宝马）、PORSCHE（保时捷）、ROLLS-ROYCE（劳斯莱斯）和 JAGUAR（美洲豹）等都采用了 CAN 总线来实现汽车内部控制系统与各检测和执行机构间的数据通信。同时，由于 CAN 总线本身的特点，其应用范围目前已不再局限于汽车行业，而向自动控制、航空航天、航海、过程工业、机械工业、纺织机械、农用机械、机器人、数控机床、医疗器械及传感器等领域发展。CAN 总线已经形成国际标准，并已被公认为几种最有前途的现场总线之一。其典型的应用协议有：SAE J1939/ISO11783、CANOpen、CANaerospace、DeviceNet、NMEA 2000 等。

（1）CAN 总线的产生与发展

控制器局部网（CONTROLLER AREA NETWORK，CAN）是 BOSCH 公司为现代汽车应用领先推出的一种多主机局部网，由于其高性能、高可靠性、实时性等优点现已广泛地应用于工业自动化、多种控制设备、交通工具、医疗仪器以及建筑、环境控制等众多部门。控制器局部网将在我国迅速普及推广。

随着计算机硬件、软件技术及集成电路技术的迅速发展，工业控制系统已成为计算机技术应用领域中最具活力的一个分支，并取得了巨大进步。由于对系统可靠性和灵活性的高要求，工业控制系统的发展主要表现为：控制面向多元化，系统面向分散化，即负载分散、功能分散、危险分散和地域分散。

分散式工业控制系统就是为适应这种需要而发展起来的。这类系统是以微型机为核心，

将 5C 技术——COMPUTER（计算机技术）、CONTROL（自动控制技术）、COMMUNICATION（通信技术）、CRT（显示技术）和 CHANGE（转换技术）紧密结合的产物。它在适应范围、可扩展性、可维护性以及抗故障能力等方面，较之分散型仪表控制系统和集中型计算机控制系统都具有明显的优越性。

典型的分散式控制系统由现场设备、接口与计算设备以及通信设备组成。现场总线（FIELDBUS）能同时满足过程控制和制造业自动化的需要，因而现场总线已成为工业数据总线领域中最为活跃的一个领域。现场总线的研究与应用已成为工业数据总线领域的热点。尽管目前对现场总线的研究尚未能提出一个完善的标准，但现场总线的高性能价格比将吸引众多工业控制系统采用。同时，正由于现场总线的标准尚未统一，也使得现场总线的应用得以不拘一格地发挥，并将为现场总线的完善提供更加丰富的依据。控制器局部网 CAN（CONTROLLER AERANETWORK）正是在这种背景下应运而生的。

由于 CAN 总线为愈来愈多不同领域采用和推广，导致要求各种应用领域通信报文的标准化。为此，1991 年 9 月 PHILIPS SEMICONDUCTORS 制订并发布了 CAN 总线技术规范（VERSION 2.0）。该技术规范包括 A 和 B 两部分。2.0A 给出了曾在 CAN 总线技术规范版本 1.2 中定义的 CAN 总线报文格式，能提供 11 位地址；而 2.0B 给出了标准的和扩展的两种报文格式，提供 29 位地址。此后，1993 年 11 月 ISO 正式颁布了道路交通运载工具——数字信息交换——高速通信控制器局部网（CAN）国际标准（ISO11898），为控制器局部网标准化、规范化推广铺平了道路。

（2）CAN 总线特点

CAN 总线是德国 BOSCH 公司从 20 世纪 80 年代初为解决现代汽车中众多的控制与测试仪器之间的数据交换而开发的一种串行数据通信协议，它是一种多主总线，通信介质可以是双绞线、同轴电缆或光导纤维。通信速率可达 1MBPS。

1）CAN 总线通信接口中集成了 CAN 协议的物理层和数据链路层功能，可完成对通信数据的成帧处理，包括位填充、数据块编码、循环冗余检验、优先级判别等项工作。

2）CAN 协议的一个最大特点是废除了传统的站地址编码，而代之以对通信数据块进行编码。采用这种方法的优点可使网络内的节点个数在理论上不受限制，数据块的标识码可由 11 位或 29 位二进制数组成，因此可以定义 211 或 229 个不同的数据块，这种按数据块编码的方式，还可使不同的节点同时接收到相同的数据，这一点在分布式控制系统中非常有用。数据段长度最多为 8 个字节，可满足通常工业领域中控制命令、工作状态及测试数据的一般要求。同时，8 个字节不会占用总线时间过长，从而保证了通信的实时性。CAN 协议采用 CRC 检验并可提供相应的错误处理功能，保证了数据通信的可靠性。CAN 总线卓越的特性、极高的可靠性和独特的设计，特别适合工业过程监控设备的互连，因此越来越受到工业界的重视，并已公认为最有前途的现场总线之一。

3）CAN 总线采用了多主竞争式总线结构，具有多主站运行和分散仲裁的串行总线以及广播通信的特点。CAN 总线上任意节点可在任意时刻主动地向网络上其他节点发送信息而不分主次，因此可在各节点之间实现自由通信。CAN 总线协议已被国际标准化组织认证，技术比较成熟，控制的芯片已经商品化，性价比高，特别适用于分布式测控系统之间的数据通信。CAN 总线插卡可以任意插在 PC AT XT 兼容机上，方便地构成分布式监控系统。

4）结构简单，只有 2 根线与外部相连，并且分布集成了错误探测和管理模块。

（3）CAN 总线技术

1）位仲裁：要对数据进行实时处理，就必须将数据快速传送，这就要求数据的物理传输通路有较高的速度。在几个站同时需要发送数据时，要求快速地进行总线分配。实时处理通过网络交换的紧急数据有较大的不同。一个快速变化的物理量，如汽车引擎负载，将比类似汽车引擎温度这样相对变化较慢的物理量更频繁地传送数据，并要求更短地延时。

CAN 总线以报文为单位进行数据传送，报文的优先级结合在 11 位标识符中，具有最低二进制数的标识符有最高的优先级。这种优先级一旦在系统设计时被确立后就不能再被更改。总线读取中的冲突可通过位仲裁解决。当几个站同时发送报文时，站 1 的报文标识符为 011111，站 2 的报文标识符为 0100110，站 3 的报文标识符为 0100111。所有标识符都有相同的两位 01，直到第 3 位进行比较时，站 1 的报文被丢掉，因为它的第 3 位为高，而其他两个站的报文第 3 位为低。站 2 和站 3 报文的 4、5、6 位相同，直到第 7 位时，站 3 的报文才被丢失。注意，总线中的信号持续跟踪最后获得总线读取权的站的报文。在此例中，站 2 的报文被跟踪。这种非破坏性位仲裁方法的优点在于，在网络最终确定哪一个站的报文被传送以前，报文的起始部分已经在网络上传送了。所有未获得总线读取权的站都成为具有最高优先权报文的接收站，并且不会在总线再次空闲前发送报文。

CAN 总线具有较高的效率是因为总线仅仅被那些请求总线悬而未决的站利用，这些请求是根据报文在整个系统中的重要性按顺序处理的。这种方法在网络负载较重时有很多优点，因为总线读取的优先级已被按顺序放在每个报文中了，这可以保证在实时系统中较低的个体隐伏时间。

对于主站的可靠性，由于 CAN 总线协议执行非集中化总线控制，所有主要通信，包括总线读取（许可）控制，在系统中分几次完成。这是实现有较高可靠性的通信系统的惟一方法。

2）CAN 总线与其他通信方案的比较：在实践中，有两种重要的总线分配方法：按时间表分配和按需要分配。在第一种方法中 ，不管每个节点是否申请总线，都对每个节点按最大期间分配。由此，总线可被分配给每个站并且是惟一的站，而不论其是立即进行总线存取或在一特定时间进行总线存取。这将保证在总线存取时有明确的总线分配。在第二种方法中，总线按传送数据的基本要求分配给一个站 ，总线系统按站希望的传送分配（如：Ethernet CSMA/CD）。因此，当多个站同时请求总线存取时，总线将终止所有站的请求，这时将不会有任何一个站获得总线分配。为了分配总线，多于一个总线存取是必要的。

CAN 实现总线分配的方法，可保证当不同的站申请总线存取时，明确地进行总线分配。这种位仲裁的方法可以解决当两个站同时发送数据时产生的碰撞问题。不同于 Ethernet 网络的消息仲裁，CAN 总线的非破坏性解决总线存取冲突的方法，确保在不传送有用消息时总线不被占用。甚至当总线在重负载情况下，以消息内容为优先的总线存取也被证明是一种有效的系统。虽然总线的传输能力不足，所有未解决的传输请求都按重要性顺序来处理。在 CSMA/CD 这样的网络中，如 Ethernet，系统往往由于过载而崩溃，而这种情况在 CAN 总线中不会发生。

3）CAN 的报文格式：在总线中传送的报文，每帧由 7 部分组成。CAN 总线协议支持两种报文格式，其惟一的不同是标识符（ID）长度不同，标准格式为 11 位，扩展格式为 29 位。

在标准格式中，报文的起始位称为帧起始（SOF），然后是由11位标识符和远程发送请求位（RTR）组成的仲裁场。RTR位标明是数据帧还是请求帧，在请求帧中没有数据字节。

控制场包括标识符扩展位（IDE），指出是标准格式还是扩展格式。它还包括一个保留位（ro），为将来扩展使用。它的最后4个字节用来指明数据场中数据的长度（DLC）。数据场范围为0~8个字节，其后有一个检测数据错误的循环冗余检查（CRC）。

应答场（ACK）包括应答位和应答分隔符。发送站发送的这两位均为隐性电平（逻辑1），这时正确接收报文的接收站发送主控电平（逻辑0）覆盖它。用这种方法，发送站可以保证网络中至少有一个站能正确接收到报文。

报文的尾部由帧结束标出。在相邻的两条报文间有一很短的间隔位，如果这时没有站进行总线存取，总线将处于空闲状态。

4）数据错误检测：不同于其他总线，CAN协议不能使用应答信息。事实上，它可以将发生的任何错误用信号发出。CAN协议可使用五种检查错误的方法，其中前三种为基于报文内容检查。

① 循环冗余检查（CRC）：在一帧报文中加入冗余检查位可保证报文正确。接收站通过CRC可判断报文是否有错。

② 帧检查：这种方法通过位场检查帧的格式和大小来确定报文的正确性，用于检查格式上的错误。

③ 应答错误：如前所述，被接收到的帧由接收站通过明确的应答来确认。如果发送站未收到应答，那么表明接收站发现帧中有错误，也就是说，ACK场已损坏或网络中的报文无站接收。CAN协议也可通过位检查的方法探测错误。

④ 总线检测：有时CAN中的一个节点可监测自己发出的信号。因此，发送报文的站可以观测总线电平并探测发送位和接收位的差异。

⑤ 位填充：一帧报文中的每一位都由不归零码表示，可保证位编码的最大效率。然而，如果在一帧报文中有太多相同电平的位，就有可能失去同步。为保证同步，同步沿用位填充产生。产生五个位。在五个连续相等位后，发送站自动插入一个与之互补的补码位；接收时，这个填充位被自动丢掉。例如，五个连续的低电平位后，CAN自动插入一个高电平位。CAN通过这种编码规则检查错误，如果在一帧报文中有6个相同位，CAN就知道发生了错误。

如果至少有一个站通过以上方法探测到 一个或多个错误，它将发送出错标志终止当前的发送。这可以阻止其他站接收错误的报文，并保证网络上报文的一致性。当大量发送数据被终止后，发送站会自动地重新发送数据。作为规则，在探测到错误后23个位周期内重新开始发送。在特殊场合，系统的恢复时间为31个位周期。

但这种方法存在一个问题，即一个发生错误的站将导致所有数据被终止，其中也包括正确的数据。因此，如果不采取自监测措施，总线系统应采用模块化设计。为此，CAN协议提供一种将偶然错误从永久错误和局部站失败中区别出来的办法。这种方法可以通过对出错站统计评估来确定一个站本身的错误并进入一种不会对其他站产生不良影响的运行方法来实现，即站可以通过关闭自己来阻止正常数据因被错误地当成不正确的数据而被终止。

⑥ CAN 可靠性：为防止汽车在使用寿命期内由于数据交换错误而对司机造成危险，汽车的安全系统要求数据传输具有较高的安全性。如果数据传输的可靠性足够高，或者残留下来的数据错误足够低的话，这一目标不难实现。从总线系统数据的角度看，可靠性可以理解为，对传输过程产生的数据错误的识别能力。

残余数据错误的概率可以通过对数据传输可靠性的统计测量获得。它描述了传送数据被破坏和这种破坏不能被探测出来的概率。残余数据错误概率必须非常小，使其在系统整个寿命周期内，按平均统计时几乎检测不到。计算残余错误概率要求能够对数据错误进行分类，并且数据传输路径可由一模型描述。如果要确定 CAN 总线的残余错误概率，我们可将残留错误的概率作为具有 80～90 位的报文传送时位错误概率的函数，并假定这个系统中有 5～10 个站，并且错误率为 1/1000，那么最大位错误概率为 10～13 数量级。例如，CAN 总线网络的数据传输率最大为 1Mbit/s，如果数据传输能力仅使用 50%，那么对于一个工作寿命 4000h、平均报文长度为 80 位的系统，所传送的数据总量为 9×1010。在系统运行寿命期内，不可检测的传输错误的统计平均小于 10～2 量级。换句话说，一个系统按每年 365 天，每天工作 8h，每秒错误率为 0.7 计算，那么按统计平均，每 1000 年才会发生一个不可检测的错误。

（4）应用举例

某医院现有 5 台 16T/H 德国菲斯曼燃气锅炉，向洗衣房、制剂室、供应室、生活用水、暖气等设施提供 5kg/cm^2 的蒸汽，全年耗用天然气 1200 万 m^3，耗用 20 万 t 自来水。医院采用接力式方式供热，对热网进行地域性管理，分四大供热区。其中冬季暖气的用气量很大，据此设计了基于 CAN 现场总线的分布式锅炉蒸汽热网智能监控系统。现场应用表明：该楼宇自动化系统具有抗干扰能力强，现场组态容易，网络化程度高，人机界面友好等特点。

CAN 总线有如下基本特点：

1）废除传统的站地址编码，代之以对通信数据块进行编码，可以多主方式工作。

2）采用非破坏性仲裁技术，当两个节点同时向网络上传送数据时，优先级低的节点主动停止数据发送，而优先级高的节点可不受影响继续传输数据，有效地避免了总线冲突。

3）采用短帧结构，每一帧的有效字节数为 8 个，数据传输时间短，受干扰的概率低，重新发送的时间短。

4）每帧数据都有 CRC 校验及其他检错措施，保证了数据传输的高可靠性，适于在高干扰环境下使用。

5）节点在错误严重的情况下，具有自动关闭总线的功能，切断它与总线的联系，以使总线上其他操作不受影响。

6）可以点对点，一对多及广播集中方式传送和接受数据。

6. MCP2515 芯片

1）MCP 2515 芯片完全支持 CAN V2.0B 技术规范，通信速率为 1 Mbit/s；

0～8 B 长的数据字段。

– 标准和扩展数据帧及远程帧

2）接收缓冲器、验收屏蔽寄存器和验收滤波寄存器：

- 两个接收缓冲器，可优先存储报文
- 6 个 29 位验收滤波寄存器
- 2 个 29 位验收屏蔽寄存器

3）对头两个数据字节进行滤波（针对标准数据帧）

4）3 个发送缓冲器，具有优先级设定及发送中止功能

5）高速 SPI 接口（10 MHz）：

- 支持 0，0 和 1，1 的 SPI 模式

6）单触发模式确保报文发送只尝试一次

7）带有可编程预分频器的时钟输出引脚：

- 可用作其他器件的时钟源

8）可用起始帧信号（Start-of-Frame，SOF），用于监控 SOF 信号：

- 可用于时隙协议和/或总线诊断以检测早期总线性能退化

9）带有可选使能设定的中断输出引脚

10）“缓冲器满”输出引脚可配置为：

- 各接收缓冲器的中断引脚
- 通用数字输出引脚

11）“请求发送（Request-to-Send，RTS）”输入引脚可各自配置为：

- 各发送缓冲器的控制引脚，用于请求立即发送报文
- 通用数字输入引脚

12）低功耗的 CMOS 技术：

- 工作电压范围 2.7 ~5.5V
- 5mA 典型工作电流
- 1μA 典型待机电流（休眠模式）

13）工作温度范围：

- 工业级（I）：-40 ~ +85℃
- 扩展级（E）：-40 ~ +125℃

18 引脚 PDIP/SOIC 如图 11-40 所示。

7. MCP2551 芯片

MCP2551 是一个可容错的高速 CAN 总线器件，可作为 CAN 总线协议控制器和物理总线接口。MCP2551 可为 CAN 总线协议控制器提供差分收发能力，它完全符合 ISO-11898 标准，包括能满足 24V 电压要求。它的工作速率高达 1 Mb/s。

在典型情况下，CAN 总线系统上的每个节点都必须有一个器件，把 CAN 总线控制器生成的数字信号转化成为适合总线传输（差分输出）的信号。它也为 CAN 总线控制器和 CAN 总线上的高压尖峰信号之间加入了缓冲器，这些高压尖峰信号可能是由外部器件产生（EMI、ESD 和电气瞬态等）。

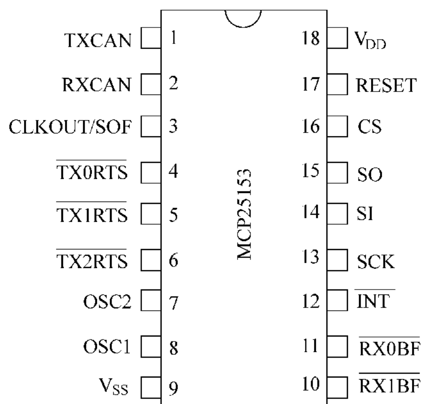


图 11-40　MCP 2515 芯片

1）支持 1 Mb/s 的运行速率；

2）满足 ISO-11898 标准物理层要求；

3）适合 12V 和 24V 系统；

4）斜率外部控制，减少 RFI；

5）自动检测 TXD 输入端的接地错误（恒显性）；

6）上电复位和电压事件欠电压保护；

7）未上电节点或欠电压不会影响 CAN 总线；

8）低电流待机操作；

9）短路保护（正负电池电压）；

10）高压瞬态保护；

11）自动热关断保护；

12）可连接节点高达 112 个；

13）采用差分总线，具有很强的抗噪声特性；

14）温度范围：

– 工业级（I）：–40 ~ +85℃

– 扩展级（E）：–40 ~ +125℃

PDIP/SOIC 如图 11-41 所示。

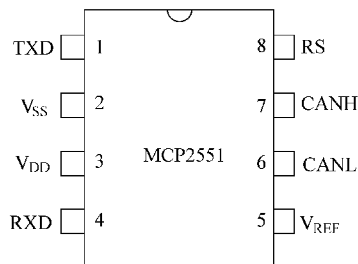


图 11-41　MCP 2551 芯片

8. RS485 总线

由于 RS232-C 接口标准出现较早，难免有不足之处，主要有以下 4 点：

1）接口的信号电平值较高，易损坏接口电路的芯片，又因为与 TTL 电平不兼容故需使用电平转换电路方能与 TTL 电路连接。

2）传输速率较低，在异步传输时，波特率为 20Kbit/s。

3）接口使用一根信号线和一根信号返回线而构成共地的传输形式，这种共地传输容易产生共模干扰，所以抗噪声干扰性弱。

4）传输距离有限，最大传输距离标准值为 50 英尺，实际上也只能 用在 50m 左右。

针对 RS232-C 的不足，于是就不断出现了一些新的接口标准，RS485 就是其中之一，它具有以下特点：

1）RS485 的电气特性：逻辑“1”以两线间的电压差为 +（2 ~ 6）V 表示；逻辑“0”以两线间的电压差为 –（2 ~ 6）V 表示。接口信号电平比 RS232-C 降低了，就不易损坏接口电路的芯片，且该电平与 TTL 电平兼容，可方便与 TTL 电路连接。

2）RS485 的数据最高传输速率为 10Mbit/s。

3）RS485 接口是采用平衡驱动器和差分接收器的组合，抗共模干能力增强，即抗噪声干扰性好。

4）RS485 最大的通信距离约为 1219m，最大传输速率为 10Mbit/s，传输速率与传输距离成反比，在 100Kbit/s 的传输速率下，才可以达到最大的通信距离，如果需传输更长的距离，需要加 485 中继器。RS485 总线一般最大支持 32 个节点，如果使用特制的 485 芯片，可以达到 128 个或者 256 个节点，最大的可以支持到 400 个节点。

因 RS485 接口具有良好的抗噪声干扰性，长的传输距离和多站能力等上述优点就使其

成为首选的串行接口。因为RS485接口组成的半双工网络，一般只需两根连线，所以RS485接口均采用屏蔽双绞线传输。RS485接口连接器采用DB-9的9芯插头座，与智能终端RS485接口采用DB-9（孔），与键盘连接的键盘接口RS485采用DB-9（针）。

5）采用RS485接口时，传输电缆的长度如何考虑?

在使用RS485接口时，对于特定的传输线经，从发生器到负载其数据信号传输所允许的最大电缆长度是数据信号速率的函数，这个长度数据主要是受信号失真及噪声等影响所限制。最大电缆长度与信号速率的关系曲线是使用24AWG铜芯双绞电话电缆（线径为0.51mm），线间旁路电容为52.5PF/M，终端负载电阻为100Ω时所得出。（曲线引自GB11014-89附录A）。当数据信号速率降低到90Kbit/s以下时，假定最大允许的信号损失为6dBV时，则电缆长度被限制在1200m。实际上，在实用时是完全可以取比它长的电缆。当使用不同线径的电缆时，则取得的最大电缆长度是不相同的。例如：当数据信号速率为600Kbit/s时，采用24AWG电缆时，大电缆长度为200m；若采用19AWG电缆（线径为0.91mm）时，则电缆长度可以大于200m；若采用28AWG电缆（线径为0.32mm）时，则电缆长度只能小于200m。

RS485总线，在要求通信距离为几十米到上千米时，广泛采用RS485串行总线标准。RS485采用平衡发送和差分接收，因此具有抑制共模干扰的能力。加上总线收发器具有高灵敏度，能检测低至200mV的电压，故传输信号能在千米以外得到恢复。RS485采用半双工工作方式，任何时候只能有一点处于发送状态，因此发送电路须由使能信号加以控制。RS485用于多点互连时非常方便，可以省掉许多信号线。应用RS485可以联网构成分布式系统，其允许最多并联32台驱动器和32台接收器。

以往，PC与智能设备通信多借助RS232、RS485、以太网等方式，主要取决于设备的接口规范。但RS232、RS485只能代表通信的物理介质层和链路层，如果要实现数据的双向访问，就必须自己编写通信应用程序，但这种程序多数都不符合ISO/OSI的规范，只能实现较单一的功能，适用于单一设备类型，程序不具备通用性。在RS232或RS485设备联成的设备网中，如果设备数量超过2台，就必须使用RS485做通信介质，RS485网的设备间要想互通信息只有通过“主（Master）”设备中转才能实现，这个主设备通常是PC，而这种设备网中只允许存在一个主设备，其余全部是从（Slave）设备。而现场总线技术是以ISO/OSI模型为基础的，具有完整的软件支持系统，能够解决总线控制、冲突检测、链路维护等问题。

9. MAX485芯片

MAX481、MAX483、MAX485、MAX487-MAX491和MAX1487芯片是用于RS485与RS422通信的低功耗收发器，每个器件中都具有一个驱动器和一个接收器。MAX483、MAX487、MAX488和MAX489芯片具有限摆率驱动器，可以减小EMI，并降低由不恰当的终端匹配电缆引起的反射，实现最高250Kbit/s的无差错数据传输。MAX481、MAX485、MAX490、MAX491、MAX1487的驱动器摆率不受限制，可以实现最高2.5Mbit/s的传输速率。这些收发器在驱动器禁用的空载或满载状态下，吸取的电源电流在120~500A之间。另外，MAX481、MAX483与MAX487具有低电流关断模式，仅消耗0.1μA。所有器件都工作在5V单电源下。

驱动器具有短路电流限制，并可以通过热关断电路将驱动器输出置为高阻状态，防止过度的功率损耗。接收器输入具有失效保护特性，当输入开路时，可以确保逻辑高电平输出。

芯片外观如图 11-42 所示。

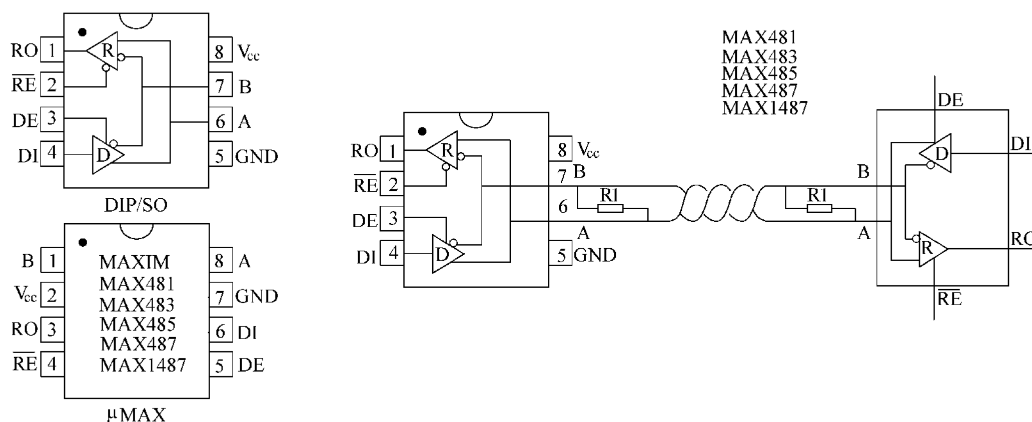


图 11-42　芯片

MAX485 通信程序与 MAX232 通信程序在本质上是一样的，只是 MAX485 通信程序需要加上通信方向控制，RS232 串口通信程序相关资料请见本书前面部分内容介绍。

11.7　KC-201 FM 立体声收音模块

1. 功能描述

KC-201 为 FM 立体声收音模块，配合单片机最小系统电路或本公司单片机实验板能够完成 88～108MHz 频率的调频广播接收。模块采用 TEA5767/CL5767 专用 FM 芯片，接收稳定可靠。板载 TDA2822 功放电路，同时有左右声道音量调节电位器。KC-201 立体声收音模块如图 11-43 所示。

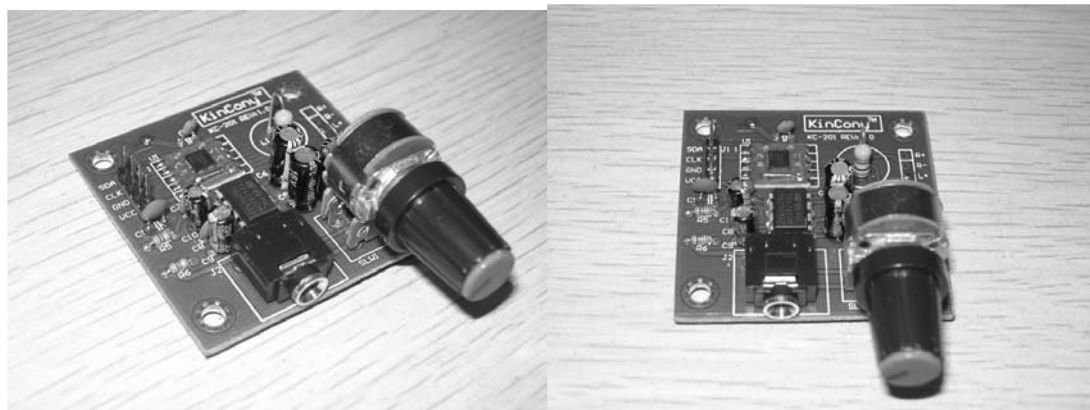


图 11-43　KC-201 FM 立体声收音模块

2. 模块主要参数和特点

1）采用通用的 102BC 模块的封装。

2）兼容 TEA5767 及 CL3767 软、硬件（软件搜索使用 IF 中频功率判断方式，如原机型使用立体声判断方式的只需少做软件改动即可）。

3）灵敏度高、噪声小、抗干扰能力强、外接元器件极少、使用极其简单，板子尺寸为 47mm×48mm 。

4）I^2C 串行数据总线接口通信。

5）采用先进的 SEEK 硬件搜台方式，全频段搜索只需 4～5s，在幅提高搜台速度。

6）内置 LDO 调整、低功耗、超宽电压使用范围（DC2.7～5.5V）。

7）内置噪声消除、软静音、低音增强电路设计。

8）FM 及 MPX 立体声采用 DSP 处理器。

9）应用简便、成本低，性价比高。

3. 硬件接口描述

J1：是与单片机连接的 IIC 总线接口，分别为 SDA（数据）、CLK（时钟）、GND（地）、VCC（电源）。

J3：是音频输出口：R+（右声道输出线+）、R-（右声道输出线-）、L+（左声道输出线+）、L-（左声道输出线-）。

J2：为外置耳机插孔，插上耳机后，左右声道音频输出自动切换到耳机上，拔掉耳机后，左右声道输出自动切换到 J3 输出端。

4. 电路原理与元器件分布

电路原理和元器件分布如图 11-44 所示。

5. 模块相关背景知识

CL3767 是一片低功耗电调谐调频立体声收音机电路，其内部集成了中频选频和解调网络，可以做到完全免调，因此只需要很少量的小体积外围元件。CL3767HN 可以应用在欧洲、美国和日本不同的 FM 波段环境。该产品具有如下特点：

1）高灵敏度（使用低噪声射频输入放大器）；

2）兼容美国/欧洲（87.5～108 MHz）和日本（76～91MHz）调频波段；

3）预调谐接收日本电视伴音至 108 MHz；

4）高放自动增益控制（AGC）电路；

5）LC 调谐振荡用低成本固定芯片；

6）调频中频选择在内部完成，中频免调；

7）三种振荡基准频率输入 32.768kHz、13MHz、6.5MHz；

8）锁相环调谐系统；

9）由总线模式引脚来选择 I^2C 总线模式或三线模式；

10）由总线输出 7 位中频计数，由总线输出 4 位电平；

11）软静音，立体声消噪（SNC），高电平切割（HCC）；

12）软静音，立体声消噪（SNC），高电平切割（HCC）能通过总线关断；

13）免调谐立体声解码器，自动搜索调谐功能；

14）待机模式；

15）两个软件可编程端口，总线输入，输出线三态模式；

16）自动调节温度范围（在 VCCA，VCC（VCO）和 VCCD=5V）。

a)

b)

图 11-44　电路原理和元器件分布

a）电路原理　b）元器件分布

6. 软件编程

（1）I^2C-总线

I^2C 总线是通过两根 I^2C 总线是通过两根线（串行数据线和串行时钟线）来连接元器件之间通信的总线，并根据地址识别每个器件。启动总线后的第一个字节的高七位是从器件的寻址地址。

IC 的地址为 C0：1100000 。

I^2C 总线的逻辑结构：收发机。

最低位未使用。

最大低电平和最小高电平分别限定在 0. 2VCCD 和 0. 45VCCD。

总线模式（BUSMODE）引脚必须接地时工作在 I^2C-总线模式。

注：总线工作在最大时钟频率（400kHz）时，不能将IC连接到一个正工作在高时钟的总线上。

（2）数据传输数据顺序

地址，字节1，字节2，字节3，字节4，字节5（数据传送必须按顺序）。地址最低位为“0”，表示写操作到CL5767。每个字节的第七位被认为是最高位，并作为字节的第一位传送。在时钟的下降沿，数据变为有效信号。在每一字节后面的停止信号可以缩短传送时间。

在整个传输完成之前发送一个停止条件：保留的字节将包含以前的信息。如果一个字没有传送完，新的字节将被使用，但新的调谐周期不会开始。通过standby位设置，芯片可以工作在省电的待机模式，总线仍然激活。屏蔽总线界面可以减小待机电流。如果总线界面被屏蔽则程序没有待机模式，芯片维持正常工作，但已经脱离总线。软口1能够被用作调谐指示器输出，在搜台没有完成的时候，软口1输出低电平。当搜到预先设置的台或搜索完成或界定波段达到时，软口1输出高电平。当第五字节最大有效位设置为逻辑1时，锁相环的参考频率改变。调谐系统能够工作在XTAL2引脚接6.5MHz晶振。

（3）上电复位

在上电复位时，静音位置“1”，其他所有位置“0”。为了初始化集成块所有位必须重新设定。

（4）I^2C-总线协议

写模式见表11-1。

表11-1　写模式

S（1）	地址（写）	A（2）	数据位	A（2）	P（3）

注意：1：S为启动条件。
2：A为应答信号。
3：P为停止条件。

读模式见表11-2。

表11-2　读模式

S（1）	地址（读）	A（2）	数据位1

注意：1：S为启动条件。
2：A为应答信号。

IC地址位见表11-3。

表11-3　IC地址位

IC地址						模式
1	1	0	0	0	0	R/W

注：1：读或者写模式
a）0对CL5767写操作。
b）1对CL5767读操作。

（5）3-线说明3-线控制

写/读，时钟和数据线；工作在最大时钟频率为1MHz。提示：通过standby位设置，芯

片可以工作在省电的待机模式。在待机模式下芯片必须设置在写模式。在待机期间，当芯片设置为读模式时，芯片会保持数据。屏蔽总线界面可以减小待机电流。如果总线界面被屏蔽则程序没有待机模式，芯片维持正常工作，但已经脱离时钟和数据线。

（6）数据传输数据顺序

地址，字节 1，字节 2，字节 3，字节 4，字节 5（数据传送必须按顺序）。在写/读控制的上升沿可以写数据到芯片。在时钟的上升沿之前，数据必须为有效信号。当时钟为低时可改变数据信号，在时钟的上升沿时数据被写入芯片。在以开始二字节或每个字节之后，如果有新的开始信号，数据传输被停止。在写/读控制的下降沿可以从芯片读数据。当时钟为低时，写/读控制改变。在写/读控制的下降沿数据端出现第一个字节的最大有效位。在时钟下降沿移存数据，在上升沿读数据。要实现两个连续的读或写操作，写/读必须固定在最少一个时钟周期。当一个搜索调谐请求被发送时，芯片将自动开始搜索，搜索方向和搜索停止电平可以设置。当搜到一个强度等于或大于停止电平时，调谐系统停止且准备好标志位为高。在搜索期间，当一个制式已经符合时，调谐系统停止且制式标志位为高。在这种情况下准备好标志位也为高。软口 1 能够被用作调谐指示器输出，在搜台没有完成的时候，软口 1 输出低电平。当搜到预先设置的台或搜索完成或界定波段达到时，软口 1 输出高电平。当第五字节最大有效位设置为逻辑 1 时，锁相环的参考频率改变。调谐系统能够工作在 XTAL2 引脚接 6. 5MHz 晶振。

（7）上电复位

在上电复位时，静音位置“1”，其他所有位任意设置。为了初始化集成块所有位必须重新设定。

（8）写数据

3 线写数据如图 11-45 所示。

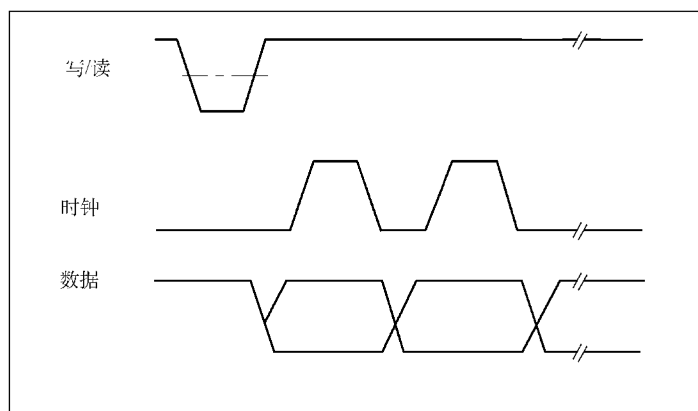


图 11-45　3 线写数据

写模式见表 11-4。

表 11-4　写模式

数据字节 1	数据字节 2	数据字节 3	数据字节 4	数据字节 5

数据字节 1 的格式见表 11-5。

表 11-5　数据字节 1 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
MUTE	SM	PLL13	PLL12	PLL11	PLL10	PLL9	PLL8

数据字节 1 的各个位描述见表 11-6。

表 11-6　数据字节 1 的各位描述

位号	符号	描　　述
7	MUTE	如果 MUTE = 1，则左右声道被静音；MUTE = 0，左右声道正常工作
6	SM	如果 SM = 1，则处于搜索模式；SM = 0，不处于搜索模式
5 到 0	PLL［13：8］	设定用于搜索和预设的可编程频率合成器

数据字节 2 的格式见表 11-7。

表 11-7　数据字节 2 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
PLL7	PLL6	PLL5	PLL4	PLL3	PLL2	PLL1	PLL0

数据字节 2 的各个位描述见表 11-8。

表 11-8　数据字节 2 的各个位描述

位号	符号	描　　述
7 到 0	PLL［7：0］	设定用于搜索和预设的可编程频率合成器

数据字节 3 的格式见表 11-9。

表 11-9　数据字节 3 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
SUD	SSL1	SSL0	HLSI	MS	ML	MR	SWP1

数据字节 3 的各个位描述见表 11-10。

表 11-10　数据字节 3 的各个位描述

位号	符号	描　　述
7	SUD	SUD = 1，增加频率搜索；SUD = 0，减小频率搜索
6 和 5	SSL［1：0］	搜索停止标准：表 11-11
4	HLSI	高/低充电电流切换：HLSI = 1，高充电电流；HLSI = 0，低充电电流
3	MS	立体声/单声道：MS = 1，单声道；MS = 0，立体声
2	ML	左声道静音：ML = 1，左声道静音并置立体声，ML = 0，左声道正常
1	MR	右声道静音：MR = 1，右声道静音并置立体声，MR = 0，右声道正常
0	SWP1	软件可编程端口 1：SWP1 = 1，端口 1 高电平；SWP1 = 0 ，端口 1 低电平

搜索停止标准设定见表 11-11。

表 11-11　搜索停止标准设定

SSL1	SSL2	搜索停止标准
0	0	在搜索模式下禁止
0	1	低：ADC 输出大小为 5
1	0	中：ADC 输出大小为 7
1	1	高：ADC 输出大小为 10

数据字节 4 的格式见表 11-12。

表 11-12　数据字节 4 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
SWP2	STBY	BL	XTAL	SMUTE	HCC	SNC	SI

数据字节 4 的各个位描述见表 11-13。

表 11-13　数据字节 4 的各个位描述

位号	符号	描　　述
7	SWP2	软件可编程端口 2：SWP2 = 1，端口 2 高电平；SWP2 = 0，端口 2 低电平
6	STBY	等待：STBY = 1，处于待机模式，STBY = 0，退出待机模式
5	BL	波段制式：BL = 1，日本调频制式；BL = 0，美国/欧洲调频制式
4	XTAL	如果 XTAL = 1 ，那么 fxtal = 32. 768kHz；如果 XTAL = 0 ，那么 fxtal = 13MHz
3	SMUTE	软件静音：SMUTE = 1，软静音打开；SMUTE = 0，软静音关闭
2	HCC	白电平切割：HCC = 1，高电平切割打开，HCC = 0，高电平切割关闭
1	SNC	立体声噪声去除：如果 SNC = 1，立体声消噪除打开，如果 SNC = 0，立体声消噪除关闭
0	SI	搜索标志位：SI = 1，SWPORT1 输出准备好信号；SI = 0，SWPORT1 作为软件可编程端口 1 用

数据字节 5 的格式见表 11-14。

表 11-14　数据字节 5 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
PLLREF	DTC						

数据字节 5 的各个位描述见表 11-15。

表 11-15　数据字节 5 的各个位描述

位号	符号	描　　述
7	PLLREF	若 PLLREF = 1，6. 5MHz 的锁相环参考频率启用；若 PLLREF = 0，6. 5MHz 的锁相环参考频率关闭
6	DTC	若 DTC = 1，去加重时间常数为 75μs；若 DTC = 0，去加重时间常数为 50μs
5 到 0		未用，状态不必考虑

（9）读数据

如图 11-46 所示。

读模式见表 11-16。

表 11-16　读模式

数据字节 1	数据字节 2	数据字节 3	数据字节 4	数据字节 5

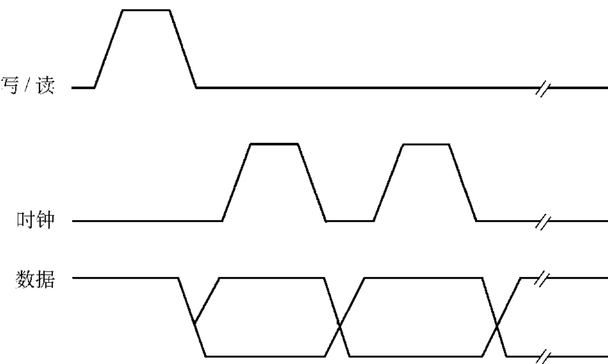


图 11-46　3 线读数据

数据字节 1 的格式见表 11-17。

表 11-17　数据字节 1 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
RF	BLF	PLL13	PLL12	PLL11	PLL10	PLL9	PLL8

数据字节 1 的各个位描述见表 11-18。

表 11-18　数据字节 1 的各个位描述

位号	符号	描　　述
7	RF	准备好标志：RF = 1，有一个频道被搜到或者一个制式已经符合；RF = 0，没有频道被搜到
6	BLF	波段制式：BLF = 1，一个制式已经符合；BLF = 0，没有制式已经符合
5 到 0	PLL［13：8］	用于搜索和预设后的可编程频率合成器设定结果

数据字节 2 的格式见表 11-19。

表 11-19　数据字节 2 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
PLL7	PLL6	PLL5	PLL4	PLL3	PLL2	PLL1	PLL0

数据字节 2 的各个位描述见表 11-20。

表 11-20　数据字节 2 的各个位描述

位号	符号	描　　述
7 到 0	PLL［7：0］	设定用于搜索和预设后的可编程频率合成器设定结果

数据字节 3 的格式见表 11-21。

表 11-21　数据字节 3 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
STEREO	IF6	IF5	IF4	IF3	IF2	IF1	IF0

数据字节 3 的各个位描述见表 11-22。

表 11-22　数据字节 3 的各个位描述

位号	符号	描　　述
7	STEREO	立体声标志位：STEREO = 1，立体声接收；STEREO = 0，单声道接收
6 到 0	IF［6：0］	中频计数器结果

数据字节 4 的格式见表 11-23。

表 11-23　数据字节 4 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
LEV3	LEV2	LEV1	LEV0	CI3	CI2	CI1	0

数据字节 4 的各个位描述见表 11-24。

表 11-24　数据字节 4 的各个位描述

位号	符号	描　述
7 到 4	LEV［3：0］	ADC 的输出
3 到 1	CI［3：1］	芯片验证号
0		该位内部置 0

数据字节 5 的格式见表 11-25。

表 11-25　数据字节 5 的格式

位 7（高位）	位 6	位 5	位 4	位 3	位 2	位 1	位 0（低位）
0	0	0	0	0	0	0	0

数据字节 5 的各个位描述见表 11-26。

表 11-26　数据字节 5 的各个位描述

位号	符号	描　述
7 到 0	—	预留为扩展用，由内部置 0

（10）总线传输时间

数字电平和传输时间见表 11-27。

表 11-27　数字电平和传输时间

符号	参数	条件	最小值	最大值	单位
数字输入					
VIH	输入高电平		0.45VCCD	—	V
VIL	输入低电平		—	0.2VCCD	V
数字输出					
Isink（L）	低电平吸收电流		500	—	μA
VOL	低电平输出电压	IOL = 500μA	—	450	mV
传输时间					
fclk	时钟输入频率	I^2C 总线	—	400	kHz
		3 线	—	1	MHz
tHIGH	时钟高电平时间	I^2C 总线	1	—	μs
		3 线	300	—	ns
tLOW	时钟低电平时间	I^2C 总线	1	—	μs
		3 线	300	—	ns
tW（write）	写操作脉冲宽度	3 线	1	—	μs
tW（read）	读操作脉冲宽度	3 线	1	—	μs
tsu（clk）	时钟建立时间	3 线	300	—	ns

（续）

符号	参数	条件	最小值	最大值	单位
传输时间					
th（out）	读操作数据输出控制时间	3 线	10	—	ns
td（out）	读操作输出延迟时间	3 线	—	100	ns
tsu（write）	写操作建立时间	3 线	100	—	ns
th（write）	写操作控制时间	3 线	100	—	ns

7. C 语言实例程序

功能：实现按键控制向上/向下频率进行搜索电台

```
sbit fm_sda = P2^2;
sbit fm_scl = P2^3;
#define ulong unsigned long
#define max_freq 108000
#define min_freq 87500

uint max_pll = 0x339b;          //108MHz 时的 pll,
uint min_pll = 0x299d;          //87.5MHz 时的 pll.

uchar radio_write_data[5] = {0x29,0x9d,0x40,0x11,0x40};     //要写入 CL5767 的数据
uchar radio_read_data[5];       //CL5767 读出的状态

ulong frequency;
uint pll;
uchar st;


void iic_start()
{
  fm_sda = 1;
  delay_us();
  fm_scl = 1;
  delay_us();
  fm_sda = 0;
  delay_us();
  fm_scl = 0;
  delay_us();
}
```

```
void iic_stop( )
{
    fm_scl = 0;
    delay_us( );
    fm_sda = 0;
    delay_us( );
    fm_scl = 1;
    delay_us( );
    fm_sda = 1;
    delay_us( );
}

void iic_ack( )
{
    fm_sda = 0;
    delay_us( );
    fm_scl = 1;
    delay_us( );
    fm_scl = 0;
    delay_us( );
    fm_sda = 1;
    delay_us( );
}


bit iic_testack( )
{
    bit ErrorBit;
    fm_sda = 1;
    delay_us( );
    fm_scl = 1;
    delay_us( );
    ErrorBit = fm_sda;
    delay_us( );
    fm_scl = 0;
    return ErrorBit;
}

void iic_write8bit( unsigned char input)
```

```
{
    unsigned char temp;
    for(temp=8;temp!=0;temp--)
    {
        fm_sda=(bit)(input&0x80);
        delay_us();
        fm_scl=1;
        delay_us();
        fm_scl=0;
        delay_us();
        input=input<<1;
    }
}

unsigned char iic_read8bit()
{
    unsigned char temp,rbyte=0;
    for(temp=8;temp!=0;temp--)
    {
        fm_scl=1;
        delay_us();
        rbyte=rbyte<<1;
        rbyte=rbyte|((unsigned char)fm_sda);
        fm_scl=0;
    }
    return rbyte;
}



void radio_write(void)
{
    unsigned char i;
    iic_start();
    iic_write8bit(0xc0);        //CL5767 写地址
    if(!iic_testack())
    {
        for(i=0;i<5;i++)
        {
```

```
            iic_write8bit(radio_write_data[i]);
            iic_ack();
        }
    }
    iic_stop();
}

//由频率计算 PLL
void get_pll(void)
{
    unsigned char hlsi;
    unsigned int twpll = 0;
    hlsi = radio_write_data[2]&0x10;
    if (hlsi)
        pll = (unsigned int)((float)((frequency + 225) * 4)/(float)32.768);    //频率单位:kHz
    else
        pll = (unsigned int)((float)((frequency-225) * 4)/(float)32.768);    //频率单位:kHz
}


//由 PLL 计算频率
void get_frequency(void)
{
    unsigned char hlsi;
    unsigned int npll = 0;
    npll = pll;
    hlsi = radio_write_data[2]&0x10;
    if (hlsi)
        frequency = (unsigned long)((float)(npll) * (float)8.192-225);    //频率单位:kHz
    else
        frequency = (unsigned long)((float)(npll) * (float)8.192 + 225);    //频率单位:kHz
}

//读 CL5767 状态,并转换成频率
void radio_read(void)
{
    unsigned char i;
    unsigned char temp_l,temp_h;
    pll = 0;
```

```
    iic_start();
    iic_write8bit(0xc1);        //CL5767 读地址
    if(! iic_testack())
    {
        for(i=0;i<5;i++)
        {
            radio_read_data[i]=iic_read8bit();
            iic_ack();
        }
    }
    iic_stop();
    temp_l=radio_read_data[1];
    temp_h=radio_read_data[0];
    st=(radio_read_data[2])&0x80;//st/mo fig
    temp_h&=0x3f;
    pll=temp_h*256+temp_l;
    get_frequency();
}

//手动设置频率,mode=1, +0.01MHz;mode=0: -0.01MHz ,不用考虑 CL5767 用于搜台的相关位:SM,SUD
void search(bit mode)
{
    radio_read();
    if(mode)
    {
        frequency+=100;
        if(frequency>max_freq)
                frequency=min_freq;
    }
    else
    {
        frequency-=100;
        if(frequency<min_freq)
                frequency=max_freq;
    }
    get_pll();
    radio_write_data[0]=pll/256;
    radio_write_data[1]=pll%256;
```

```
    radio_write_data[2] =0x41;
    radio_write_data[3] =0x11;
    radio_write_data[4] =0x40;
    radio_write();
}

//自动搜台,mode =1,频率增加搜台;mode =0:频率减小搜台
void auto_search(bit mode)
{
    radio_read();
    if(mode)
    {
        radio_write_data[2] =0xb1;
        frequency + =20;
        if(frequency > max_freq)
                frequency = min_freq;
    }
    else
    {
        radio_write_data[2] =0x41;
        frequency - =20;
        if(frequency < min_freq)
                frequency = max_freq;
    }
    get_pll();

    radio_write_data[0] = pll/256 +0x40;       //加 0x40 是将 SM 置为 1 为自动搜索模式
    radio_write_data[1] = pll%256;
    radio_write_data[3] =0x11;                 //SSL1 和 SSL0 控制搜索停止条件
    radio_write_data[4] =0x40;
    radio_write();
    radio_read();
    while(!(radio_read_data[0]&0x80))          //搜台成功标志
    {
        radio_read();
    }
}
void display(ulong date)
{
```

```
uchar ge,shi,bai,qian=0,i;
qian=date/100000;
bai=date%100000/10000;
shi=date%100000%10000/1000;
ge=date%100000%10000%1000/100;

if(st==0x80)
{
   write_com(0x80+0x40+13);
   for(i=0;i<2;i++)
   {
      write_date(tab_9[i]);
      led2=0;
   }
}
else
{

   write_com(0x80+0x40+13);
   for(i=0;i<2;i++)
   {
      write_date(0x20);
      led2=1;
   }
}

write_com(0x80+0x40+4);
if(qian==0)
{
   write_date(0x20);
}
else
{
   write_date(0x30+qian);
}
write_date(0x30+bai);
write_date(0x30+shi);
write_date(0x2e);
write_date(0x30+ge);
```

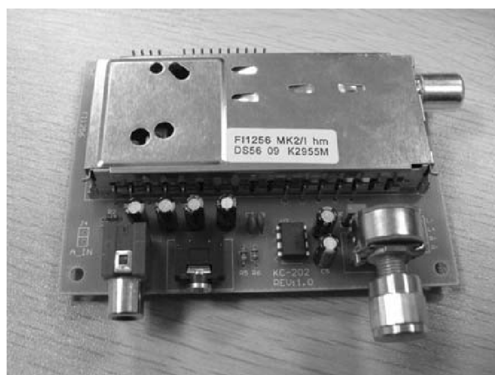
```
    write_com(0x80 +0x40 +9);
    for(i =0;i <3;i + +)
    {
        write_date(tab_10[i]);
    }
}
```

对于 select 模型，首先建立套接字集合，即 FD_SET，该集合分为可读、可写和出错三种状态；将需要的套接字加入相应集合，然后调用 select 函数，把集合和超时时间设置传入函数，如果超时时间设为 NULL，socket 则为阻塞方式工作。当 select 返回时，用 FD_ISSET 宏判断对应集合返回情况。即用一个函数处理多个套接字的通信。对于 UDP 协议，可写没用实际意义，只代表本机缓冲可用，不代表对端能够接收。所以只用 select 函数判断两个套接字的可读状态。当确定某个套接字可读时，调用 recvfrom 函数从指定 socket 接收数据。由于 UDP 是数据报协议，需保留信息边界，即发送端调用了几次 sendto，接收端就要调用几次 recvfrom，不存在一次接收到多个包的情况。所以可以直接根据接收到的数据大小判断接收到的数据包类型，然后调用对应的包处理函数，进入对应状态机。这也是所谓了 Dispatch 分包功能。这个 UDP 接收线程的工作方式和原理客户端与服务器端相同，可以理解为整个协议栈的入口。

11.8 KC-202 电视信号接收模块

1. 功能描述

KC-202 为电视信号接收模块，配合单片机最小系统电路或本公司单片机实验板能够完成48.25 ~863.25MHz 频率的电视信号接收。模块采用飞利浦 FI1256 数字式高频头，接收稳定可靠。板载 TDA2822 功放电路，同时有左右声道音量调节电位器。KC-202 电视信号接收模块如图 11-47 所示。



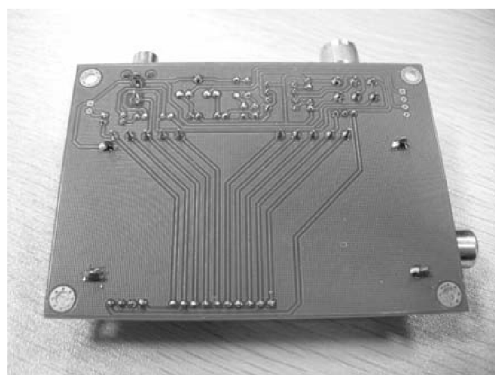


图 11-47　KC-202 电视信号接收模块

2. 模块主要参数

1）全频道调谐器。其频率范围为

48.25 ~863.25MHz，共划分为三个频段：

48.25～168.25MHz，175.25～455.25MHz，

463.25～863.25MHz。

2）锁相控制、跟踪调谐。

3）图像中频解调器带锁相控制。

4）视频输出、音频输出。

5）I^2C 总线控制频道调谐及地址选择。

6）体积小、卧式放置。它有三路输入、振荡和混频。

3. 硬件接口描述

“J1”：FI1256 高频头的各引脚端口，板上有标引脚定义。

“J2”：复合视频信号输出端，可以与电视机或监视器相连，也可以和计算机的视频采集卡相连。

“J3”：板子对外提供 5V 或扩展电源电压，VCC 为 5V，EXT_DC 为外部输入电压。

“J4”：TDA2822 功放为双声道功放，J4 为其中一路空闲的功放信号输入端，可以为用户提供声音的功率放大。

“J5”：TDA2822 功放芯片电源选择跳线，VCC 即为 TDA2822 提供 5V 电源电压，EXT_DC 为用户自己输入外接电源电压，TDA2822 的功放功率跟实际供电电压有关。

“J6”：外置耳机插孔，插上耳机后，左右声道音频输出自动切换到耳机上，拔掉耳机后，左右声道输出自动切换到 J7 输出端。

“J7”是音频输出口：R＋（右声道输出线＋）、R－（右声道输出线－）、L＋（左声道输出线＋）、L－（左声道输出线－）。

“SLW1”：音量调节电位器。

4. 电路原理与元器件分布

电路原理和元器件分布如图 11-48 所示。

5. 模块相关背景知识

（1）飞利浦 FI1256 高频头知识

FI1256 的输入级将天线来的信号输入到三个不同频段的滤波器，进行放大和滤波。输入级的滤波器用来调谐频率。它们将高频头的频率范围划分为高、中、低三个频段。输入级的滤波器受 PLL 锁相环调谐系统控制，以实现锁相调谐和频段切换。三路输入级的输出分别和相应的高、中、低三个不同频段的振荡器的输出混频。混频器输出一个中频信号，送入中频放大器进行放大。利用 I^2C 总线通信对 PLL 锁相环调谐系统编程，达到调谐和控制的目的。如图 11-49 所示。

中频信号经过声表面滤波器后输入到锁相控制中频解调器模块。中频解调器模块输出复合视频信号及声音信号。

高频头内部含有 AGC 控制。中频解调器模块输出的 AGC 信号反馈至输入级的放大控制电路，AGC 控制输入级的放大量，这样就可以使中频解调器输出稳定的电平。

（2）引脚功能

外观图如图 11-50 所示。功能说明如图 11-51 所示。

高频头引脚功能如下：VT（11 脚）：供高频调谐器用的＋33V 电源，通过 22kΩ 外接电阻连接至该引脚；＋5V Vb（12 脚）：高频头的＋5V 电源；SCL（13 脚）：I^2C 总线的串行

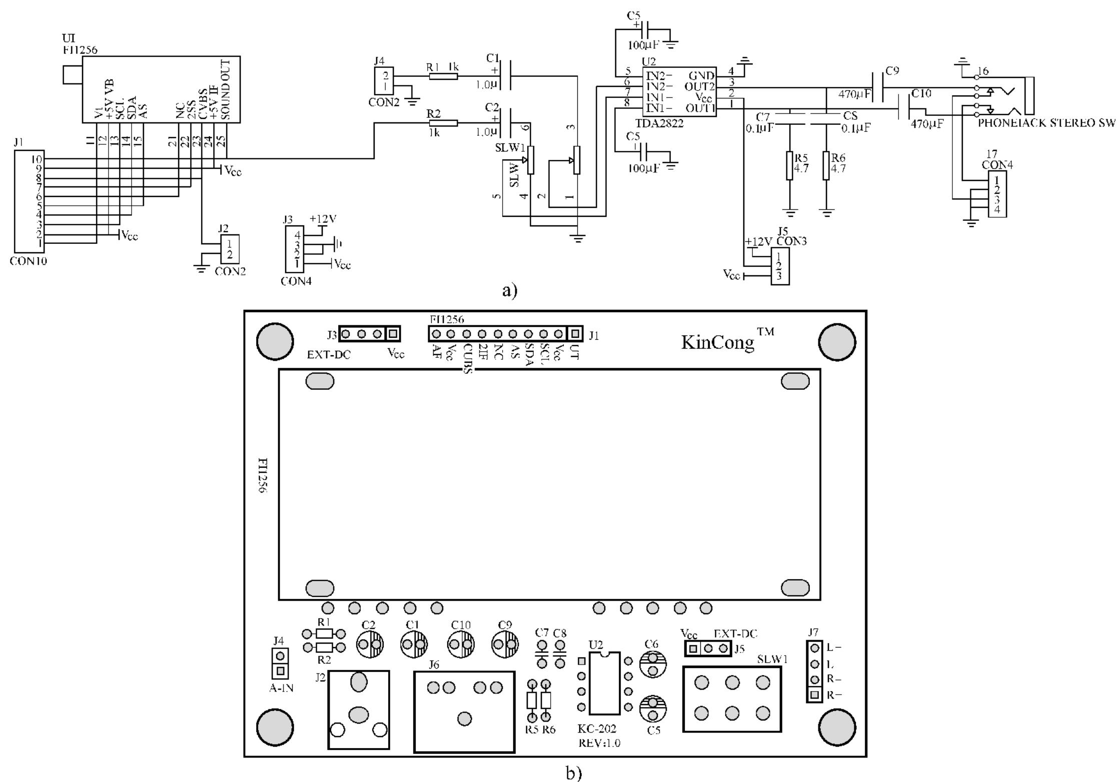


图 11-48　电路原理和元器件分布
a）电路原理　b）元器件分布

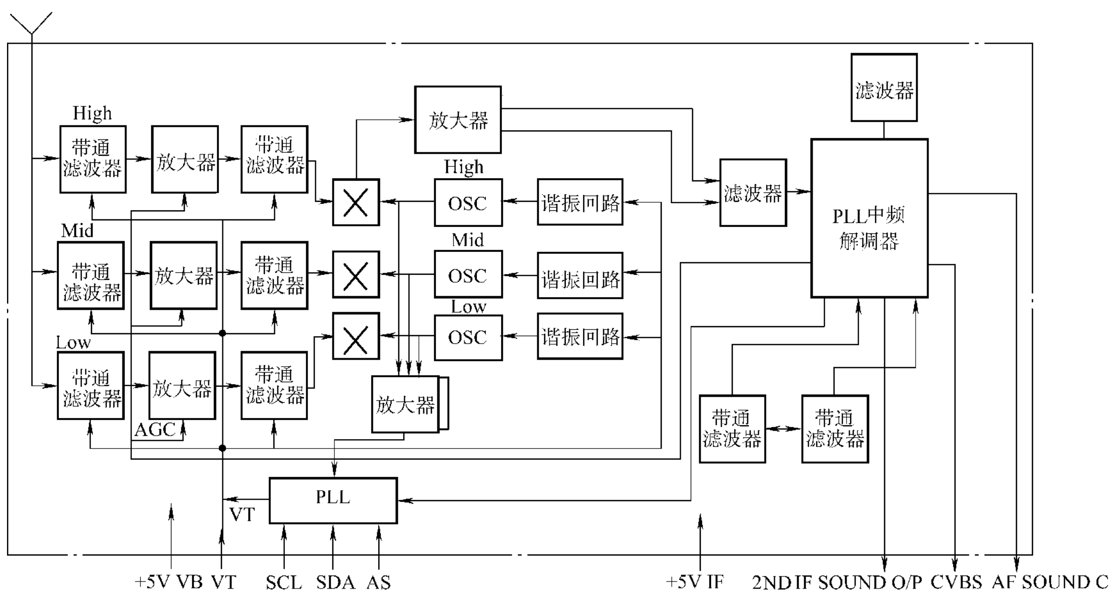
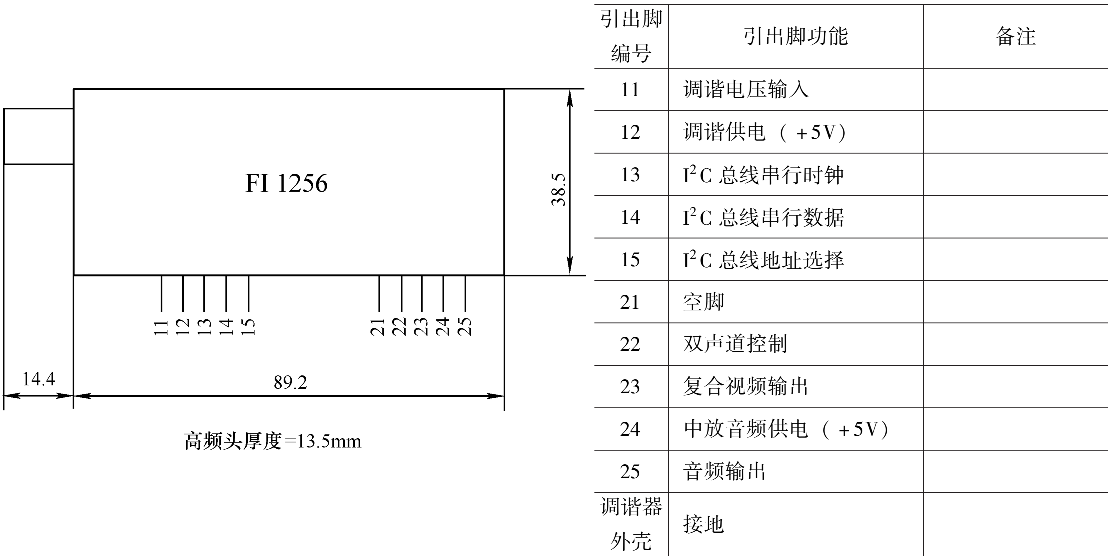


图 11-49　飞利浦 FI1256 高频头示意



图 11-50　外观图



引出脚编号	引出脚功能	备注
11	调谐电压输入	
12	调谐供电（+5V）	
13	I^2C 总线串行时钟	
14	I^2C 总线串行数据	
15	I^2C 总线地址选择	
21	空脚	
22	双声道控制	
23	复合视频输出	
24	中放音频供电（+5V）	
25	音频输出	
调谐器外壳	接地	

图 11-51　功能说明

时钟线；SDA（14 脚）：I^2C 总线的串行数据线；AS（15 脚）：I^2C 总线的地址选择线；CVBS（23 脚）：复合视频信号输出；+5V IF（24 脚）：中频解调器模块的+5V 电源；AF SOUND O/P（25 脚）：声音信号输出。

（3）编程要点

该高频头是利用 I^2C 总线对其内部寄存器编程实现各种控制功能的。

高频头的地址编码为 ADR（1 1 0 0 0 MA1 MA0 R/ W）。通常取 MA1 =0、MA0 =1。R/W = 0 时为写模式，当 R/W =1 时为读模式。高频头的控制字 CW1、CW2 定义为：CW1（1CP

T1T01110S）、CW2（P7 P6 P5 P4 P3 P2 P1P0）。通常取 T1 =0、T0 =0。当 CP =1 时为快速调谐方式，当 CP = 0 时为中速调谐方式。OS =0 时为普通工作方式，当 OS =1 时，可以通过手动调节 VT 电压以实现频率调谐。CW2 为频段切换控制字，每位的含义及设置如下：P0、P1 、P2 取值任意，当 P7 P6 P5 P4 P3 =10100 时为低频段，P7 P6 P5 P4 P3 =10010 时为中频段，当 P7 P6 P5 P4 P3 = 00110 时为高频段。高频头的分频系数 DR1、DR2 控制调谐频率的变化。DR1 的格式为：DR1（0 N14 N13 N12 N11 N10 N9 N8），DR2 的格式为（N7N6 N5 N4 N3 N2 N1 N0）。振荡频率 FOSC =N/16（MHz），N 为由 N13 ~ N0 组成的二进制数。

I^2C 总线是两线双向多主机总线。按照 I^2C 通信规则及上述编程要点对微处理器编程，即可对该高频头进行灵活控制。

我们来看一个用 51 单片机控制 FI1256 高频头输出电视信号的实验过程。我们用电视信号发生仪提供给 FI1256 高频头射频信号，用 51 单片机通过 IIC 总线方式来控制频道的输出，高频头输出的复合视频信号，通过计算机的视频采集卡输入到电脑屏幕上，这时可以看到电脑屏幕上出现了电视信号测试画面。图 11-52 ~ 图 11-61 介绍了“KC-202 电视信号接收模块”实物外观与硬件资源图片，图 11-50 ~ 图 11-61 为实验全过程演示。



图 11-52　KC-202 电视信号接收模块外观



图 11-53　KinCony 为正版产品商标

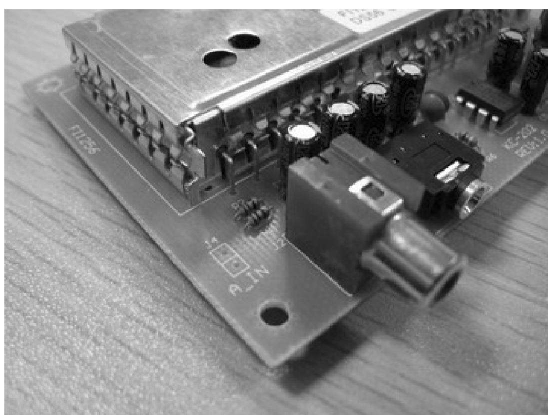


图 11-54　复合视频信号输出端



图 11-55　板载 TDA2822 功放电路（空余一路功放通道供用户自行使用）

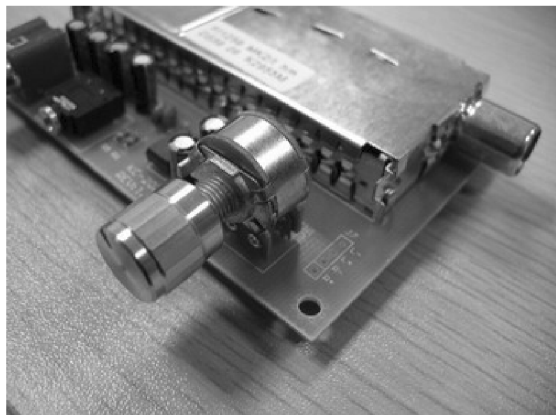


图 11-56　优质的音量调节电位器



图 11-57　天线信号输入端（可以使用无线电视天线，也可以连接有线电视信号源）



图 11-58　外扩展接口俱全（PCB 四周固定螺柱让 PCB 底层避免被金属物碰）



图 11-59　电视信号发生仪给 KC-202 模块做信号源

接收模块将收到的电视信号送入计算机视频采集卡，也可以直接接电视机的 AV 信号输入端，如图 11-61 所示。

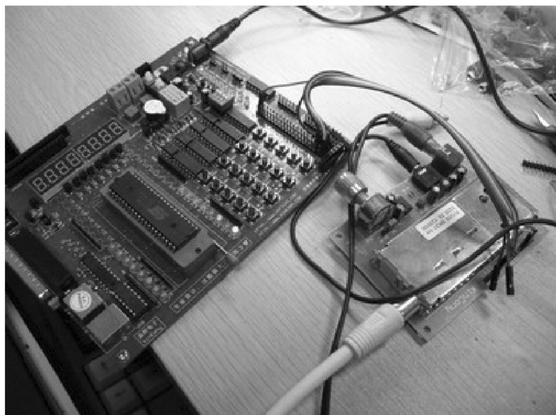


图 11-60　51 单片机综合学习系统与 KC-202 电视信号接收模块相连

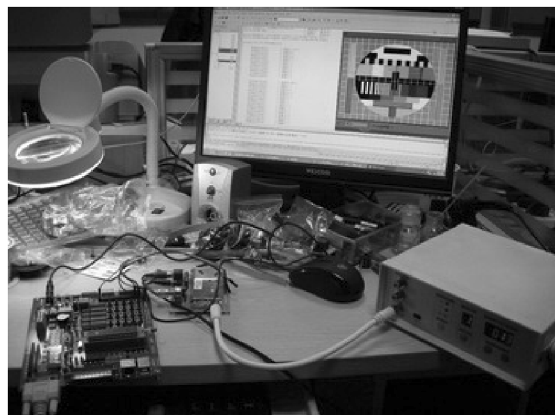


图 11-61　电视信号送入计算机视频采集卡

附录　Keil 开发软件的介绍

1. Keil 开发软件的介绍

Keil IDE（μVision2）集成开发环境是 Keil Software Inc/Keil Elektronik GmbH 开发的基于 80C51 内核的微处理器软件开发平台，内嵌多种符合当前工业标准的开发工具。可以完成从工程建立和管理、编译、连接，目标代码的生成，软件仿真，硬件仿真等完整的开发流程。尤其 C 语言的编译工具在产生代码的准确性和效率方面达到了较高的水平，而且可以附加灵活的控制选项，在开发大型项目时非常理想。

（1）μVision2 IDE：包括一个工程管理器，一个功能丰富并有交互式错误提示的编辑器、选项设置、生成工具以及在线帮助，您可以使用 μVision2 创建源文件，并组成应用工程加以管理。μVision2 可以自动完成编译、汇编、链接程序的操作，使您可以只专注开发工作的效果。

（2）C51 编译器和 A51 汇编器：由 μVision2 IDE 创建的源文件，可以被 C51 编译器或 A51 汇编器处理，生成可重定位的 object 文件。

Keil C51 编译器遵照 ANSI C 语言标准，支持 C 语言的所有标准特性。另外，还增加了几个可以直接支持 80C51 结构的特性。Keil A51 宏汇编器支持 80C51 及其派生系列的所有指令集。

（3）LIB51 库管理器：该管理器可以从由汇编器和编译器创建的目标文件建立目标库。这些库是按规定格式排列的目标模块，可在以后被链接器所使用。当链接器处理一个库时，仅仅使用了库中程序使用了的目标模块而不是全部加以引用。

（4）BL51 链接器 定位器：BL51 链接器使用从库中提取出来的目标模块和由编译器、汇编器生成的目标模块，创建一个绝对地址目标模块。绝对地址目标文件或模块包括不可重定位的代码和数据。所有的代码和数据都被固定在具体的存储器单元中。绝对地址目标文件可以用于：

1）编程 EPROM 或其他存储器设备；

2）由 μVision2 调试器对目标进行调试和模拟；

3）使用在线仿真器进行程序测试。

（5）μVision2 软件调试器：该调试器能十分理想地进行快速，可靠的程序调试。调试器包括一个高速模拟器，您可以使用它模拟整个 80C51 系统。包括片上外围器件和外部硬件。当您从器件数据库选择器件时。这个器件的属性会被自动配置。

（6）μVision2 硬件调试器：该调试器向您提供了几种在实际目标硬件上测试程序的方法。

安装 MON51 目标监控器到您的目标系统，并通过 Monitor-51 接口下载您的程序。

使用高级 GDI 接口，将 μVision2 调试器同仿真器的硬件系统相连接，通过 μVision2 的人机交互环境指挥连接的硬件完成仿真操作。

（7）RTX51 实时操作系统：该系统是针对 80C51 微控制器系列的一个多任务内核。

RTX51 实时内核简化了需要对实时事件进行反应的复杂应用的系统设计、编程和调试。这个内核完全集成在 C51 编译器中，使用非常简单。任务描述表和操作系统的一致性由 BL51 链接器/定位器自动进行控制。

2. Keil 软件开发的流程

对于刚刚使用 Keil 的用户来讲，一般是按照下面的流程来完成开发任务的：

1）建立工程。

2）为工程选择目标器件。例如选择 PHILIPS 的 P89C58。

3）设置工程的配置参数。

4）打开/建立程序文件。

5）编译和连接工程。

6）纠正程序中的书写和语法错误并重新编译连接。

7）对程序中某些纯软件的部分使用软件仿真验证。

8）使用 TKS 硬件仿真器对应用程序进行硬件仿真。

9）将生成的 Hex 文件烧写到 ROM 中运行测试。

上面的流程只是一个标准的开发流程，实际中用户可能反复重复一个或几个步骤。

3. Keil 软件的安装

本节将解释如何设置操作环境以及如何将软件安装到您的硬盘上。

（1）系统要求：必须满足最小的硬件和软件要求，才能确保编译器以及其他程序功能正常。您必须具有：

Pentium、Pentium-Ⅱ或兼容处理器的 PC；

Windows95、Windows98、Windows NT4.0、Windows2000 或 WindowsXP

至少 16MB RAM；

至少 20MB 硬盘空间。

（2）安装详细说明：所有的 Keil 产品都自带一个安装程序和安装说明，非常易于安装。根据您得到的软件途径不同，软件的存放格式可能不同。

4. Keil 软件的工作环境

安装完成后，用户可以点击运行图标进入 IDE 环境。

μVision2 软件有菜单栏，可以快速选择命令按钮的工具栏，一些源代码文件窗口，对话框窗口，信息显示窗口。μVision2 允许同时打开几个源程序文件。

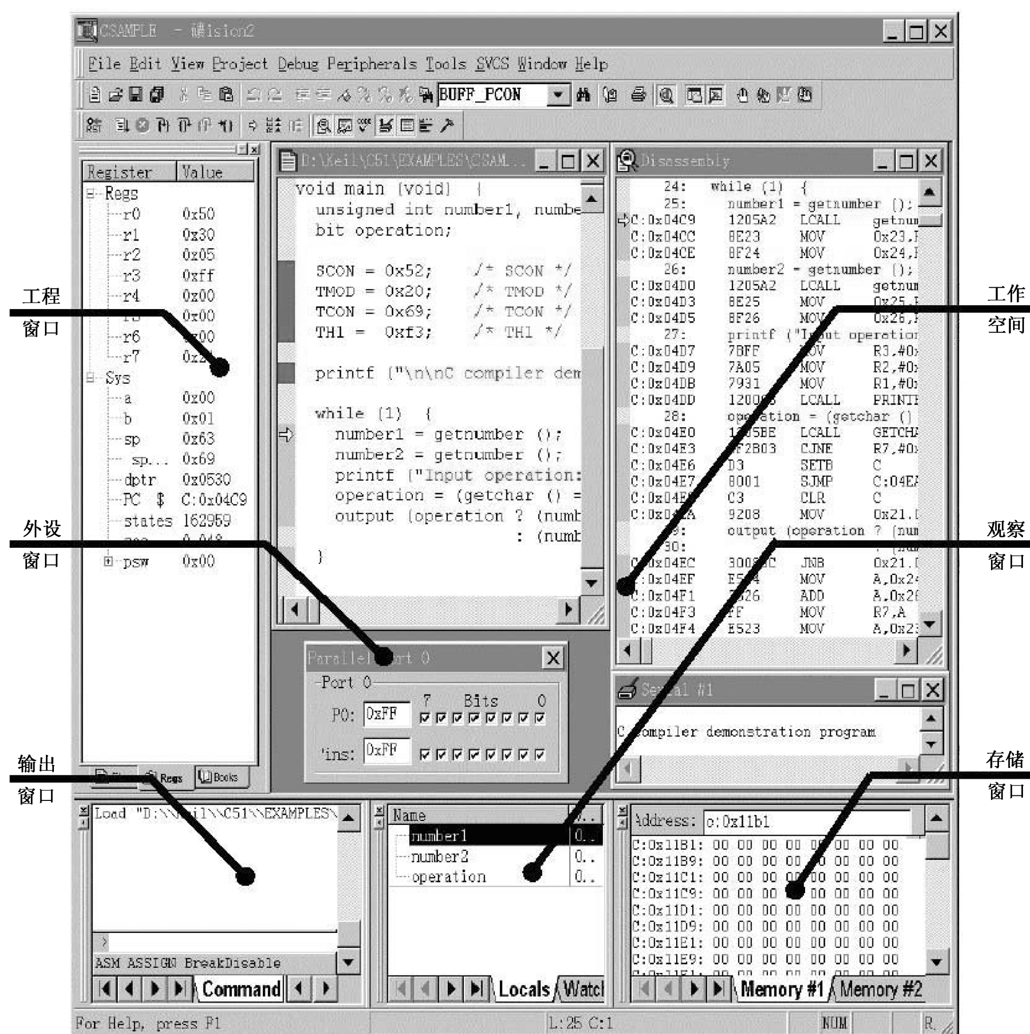
菜单栏命令、工具栏和快捷方式

菜单栏为您提供了各种操作菜单，比如：编辑器操作、工程维护、开发工具选项设置、程序调试、窗体选择和操作、在线帮助。工具栏按钮可以快速执行 μVision2 命令。快捷键（您可以自己配置）也可以执行 μVision2 命令。μVision2 的菜单项和命令、工具栏图标、默认快捷键以及它们的说明如附图 1 所示。

文件菜单和文件命令（File）如附图 2 所示。

编辑菜单和编辑器命令（Edit）如附图 3 所示。

选择文本命令：在 μVision2 中，您可以按下 Shift 键和相应的光标键来选择文字。例如：Ctrl + 是将光标移到下一个单词，而 Ctrl + Shift + 是选中从光标的位置到下一个单词开始前的文字，您也可以用鼠标选择文字。如附图 4 所示。



附图 1　μVision2 操作界面

File 菜单	工具栏	快捷键	描述
New		Ctrl+N	创建一个新的源文件或文本文件
Open		Ctrl+O	打开已有的文件
Close			关闭当前的文件
Save		Ctrl+S	保存当前的文件
			保存所有打开的源文件和文本文件
Save as…			保存并重新命名当前的文件
Device Database			维护μVision2 器件数据库
Print Setup…			设置打印机
Print		Ctrl+P	打印当前的文件
Print Preview			打印预览
1-9			打开最近使用的源文件或文本文件
Exit			退出μVision2，并提示保存文件

附图 2　文件菜单和文件命令

Edit 菜单	工具栏	快捷键	描述
		Home	将光标移到行的开始处
		End	将光标移到行的结尾处
		Crtl+Home	将光标移到文件的开始处
		Ctrl+End	将光标移到文件的结尾处
		Ctrl+←	将光标移到上一个单词
		Ctrl+→	将光标移到下一个单词
		Ctrl+A	选中当前文件中的所有文字
Undo		Ctrl+Z	撤销上一次操作
Redo		Ctrl+Shift+Z	重做上一次撤销的命令
Cut		Ctrl+X	将选中的文字剪切到剪贴板
		Ctrl+Y	将当前行的文字剪切到剪贴板
Copy		Ctrl+C	将选中的文字复制到剪贴板
Paste		Ctrl+V	粘贴剪贴板的文字
Indent Selected Text			将选中的文字向右缩进一个制表符位
Unindent Selected Text			将选中的文字向左缩进一个制表符位
Toggle Bookmark		Ctrl+F2	在当前行放置书签
Goto Next Bookmark		F2	将光标移到下一个书签
Goto Previous Bookmark		Shift+F2	将光标移到上一个书签
Clear All Bookmarks			清除当前文件中的所有书签
Find	command	Ctrl+F	在当前文件中查找文字
		F3	继续向前查找文字
		Shift+F3	继续向后查找文字
		Ctrl+F3	查找光标处（选中）的单词
		Ctrl+]	查找匹配的花括号、圆括号、方括号（使用这个命令时请将光标移到一个花括号、圆括号或方括号的前面）
Replace		Ctrl+H	替换特定的文字
Find in Files…			在几个文件中查找文字

附图 3　编辑菜单和编辑器命令

选择…	鼠标要…
任何数量的文字	在文字上拖动
一个单词	双击这个单词
一行文字	将鼠标移到行的左边，直到它变成一个指向右的箭头，然后点击
多行文字	将鼠标移到行的左边，直到它变成一个指向右的箭头，然后向上或向下拖动鼠标
垂直的一块文字	按着 Alt 键，然后拖动

附图 4　选择文本命令

视图菜单（View）如附图 5 所示。

View 菜单	工具栏	快捷键	描述
Status Bar			显示或隐藏状态栏
File Toolbar			显示或隐藏文件工具栏
Build Toolbar			显示或隐藏编译工具栏
Debug Toolbar			显示或隐藏调试工具栏
Project Window			显示或隐藏工程窗口
Output Window			显示或隐藏输出窗口
Source Browser			打开源（文件）浏览器窗口
Disassembly Window			显示或隐藏反汇编窗口
Watch & Call Stack Window			显示或隐藏观察和堆栈窗口
Memory Window			显示或隐藏存储器窗口
Code Coverage Window	CODE		显示或隐藏代码覆盖窗口
Performance Analyzer Window			显示或隐藏性能分析窗口
Symbol Window			显示或隐藏符号变量窗口
Serial Window #1			显示或隐藏串行窗口 1
Serial Window #2			显示或隐藏串行窗口 2
Toolbox			显示或隐藏工具箱
Periodic Window Update			在运行程序时，周期刷新调试窗口
Workbook Mode			显示或隐藏工作簿窗口的标签
Options…			设置颜色、字体、快捷键和编辑器选项

附图 5　视图菜单

工程菜单和工程命令（Project）如附图 6 所示。

Project 菜单	工具栏	快捷键	描述
New Project…			创建一个新的工程
Import μVision1 Project…			输入一个μVision1 工程文件
Open Project…			打开一个已有的工程
Close Project…			关闭当前的工程
Target Environment			定义工具系列、包含文件、库文件的路径
Targets，Groups，Files			维护工程的对象、文件组和文件
Select Device for Target			从器件数据库选择一个 CPU
Remove…			从工程中删去一个组或文件
Options…		Alt+F7	设置对象、组或文件的工具选项
			设置当前目标的选项
	MCB251		选择当前目标
File Extensions			选择文件的扩展名以区别不同的文件类型
Build Target		F7	转换修改过的文件并编译成应用
Rebuild Target			重新转换所有的源文件并编译成应用
Translate…		Ctrl+F7	转换当前的文件
Stop Build			停止当前的编译进程
1-9			打开最近使用的工程文件

附图 6　工程菜单和工程命令

调试菜单和调试命令（Debug）如附图 7 所示。

外围器件菜单（Peripherals）如附图 8 所示。

工具菜单（Tools）：通过工具菜单，可以配置和运行 Gimpel PC-Lint，Siemens Easy-Case 和用户程序。执行 Customize Tools Menu… 可以将用户程序添加到菜单中。如附图 9 所示。

软件版本控制系统菜单（SVCS）：这个菜单可以配置和添加软件版本控制系统（Software Version Control System）命令。如附图 10 所示。

帮助菜单（Help）如附图 11 所示。

5.“Hello”测试程序

HELLO：您的第一个 80C51 C 程序。

HELLO 示范程序位于缺省目录 C:\KEIL\C51\EXAMPLES\HELLO\中，这个程序只是将文本 Hello World 输出到串行口中，整个程序只包含一个源文件 HELLO. C，这个小型的应用程序帮助您确定工具软件可以编译、链接和调试一个应用程序。

Debug 菜单	工具栏	快捷键	描述
Start/Stop Debugging		Ctrl+F5	启动或停止μVision2 调试模式
Go		F5	运行（执行），直到下一个有效的断点
Step		F11	跟踪运行程序
Step Over		F10	单步运行程序
Step out of current function		Ctrl+F11	执行到当前函数的程序
Stop Running		ESC	停止程序运行
Breakpoints…			打开断点对话框
Insert/Remove Breakpoint			在当前行设置 / 清除断点
Enable / Disable Breakpoint			使能 / 禁能当前行的断点
Disable All Breakpoints			禁能程序中所有断点
Kill All Breakpoints			清除程序中所有断点
Show Next Statement			显示下一条执行的语句 / 指令
Enable/Disable Trace Recording			使能跟踪记录，可以显示程序运行轨迹
View Trace Records			显示以前执行的指令
Memory Map…			打开存贮器空间配置对话框
Performance Analyzer…			打开性能分析器的设置对话框
Inline Assembly…			对某一行重新汇编，可以修改汇编代码
Function Editor			编辑调试函数和调试配置文件

附图 7　调试菜单和调试命令

HELLO 的硬件是标准的，80C51 CPU 使用的唯一在片外围器件是串行口，您不需要一个实际的目标 CPU。因为 μVision2 可以模拟程序所需要的硬件。

打开 HELLO 工程文件：在 μVision 中应用程序都包含在工程中，HELLO 的工程文件已经创建好，选择 Project 菜单的 Open Project 从文件夹…\ C51 \ EXAMPLES \ HELLO 打开 HELLO. UV2 载入工程文件。如附图 12 所示。

编辑 HELLO. C：双击 Project Window 页中的 HELLO. C 现在就可以开始编辑 HELLO. C

Peripheral 菜单	工具栏	快捷键	描述
Reset CPU	RST		复位 CPU
Interrupt，I/O-Ports，Serial，Timer，A/D Converter，D/A Converter，I2C Controller，CAN Controller，Watchdog			打开在片外围器件的对话框。对话框的列表和内容由您在器件数据库中选择的 CPU 决定，不同的 CPU 会有所不同。

附图 8　外围器件菜单

Tools 菜单	工具栏	快捷键	描述
Setup PC-Lint…			配置 Gimpel Software 公司的 PC-Lint
Lint			在当前的编辑文件中运行 PC-Lint
Lint all C Source Files			在工程的 C 源代码文件中运行 PC-Lint
Setup Easy-Case…			配置 Siemens Easy-Case
Start/Stop Easy-Case			启动或停止 Siemens Easy-Case
Show File（Line）			在当前编辑的文件中运行 Easy-Case
Customize Tools Menu…			将用户程序加入工具菜单

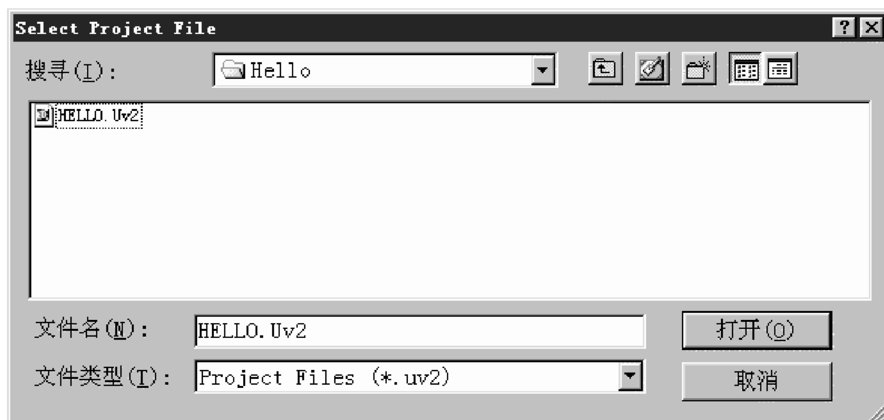
附图 9　工具菜单

SVCS 菜单	工具栏	快捷键	描述
Configure Version Control…			配置您的软件版本控制系统命令

附图 10　软件版本控制系统菜单

Help 菜单	工具栏	快捷键	描述
Help topics			打开在线帮助
About µVision			显示µVision 的版本号和许可信息

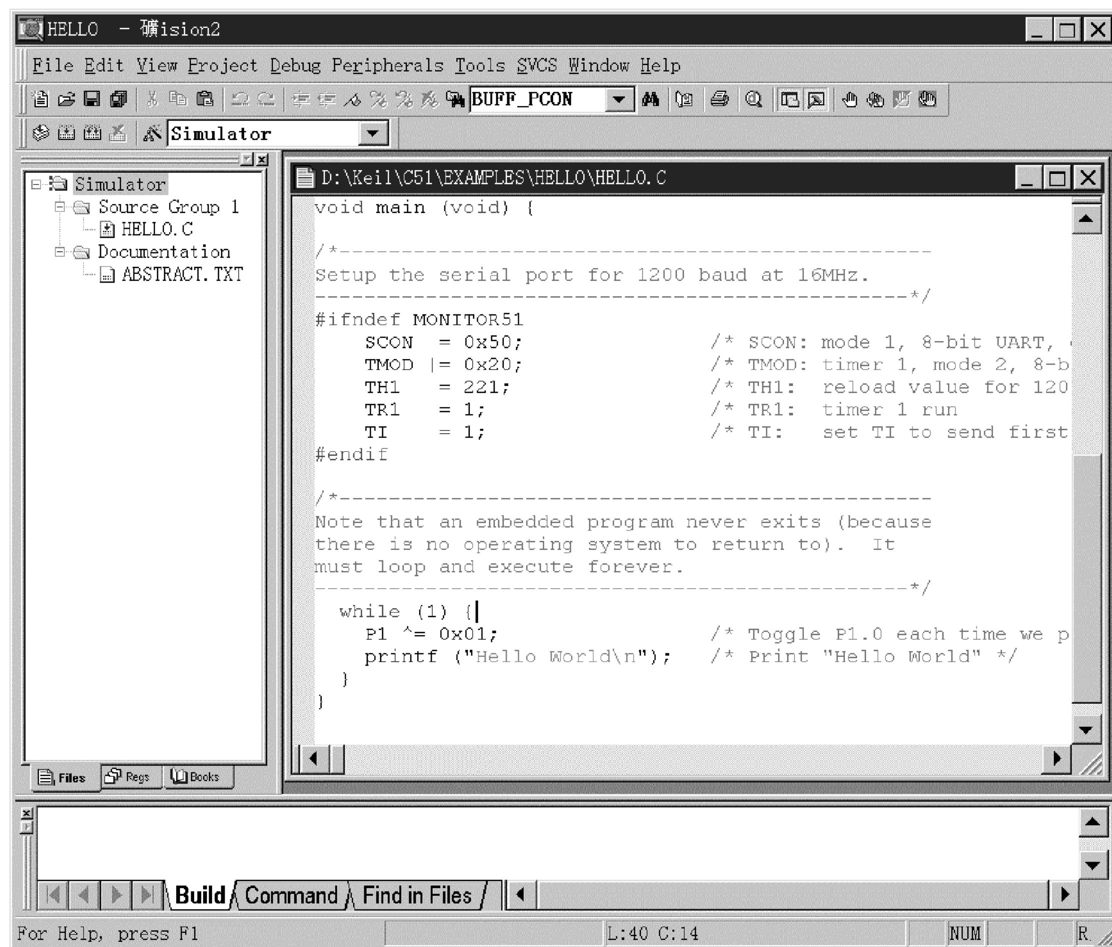
附图 11　帮助菜单



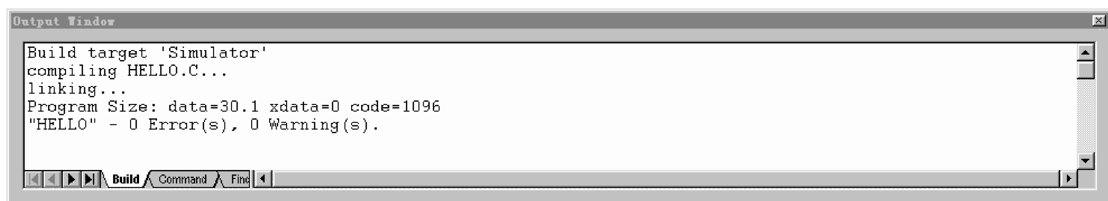
附图 12　打开 HELLO 工程文件

了，μVision2 在编辑器窗口载入和显示 HELLO. C 的内容，如附图 13 所示。

编译和链接 HELLO：用 Project 菜单或工具栏的 Build Target 命令编译和链接工程。μVision2 开始编译和链接源文件，并创建一个可以载入到 μVision2 调试器调试的绝对目标模块。编译过程的结果列在 Output Window 的 Build 页上，如附图 14 所示。注意：在 μVision2



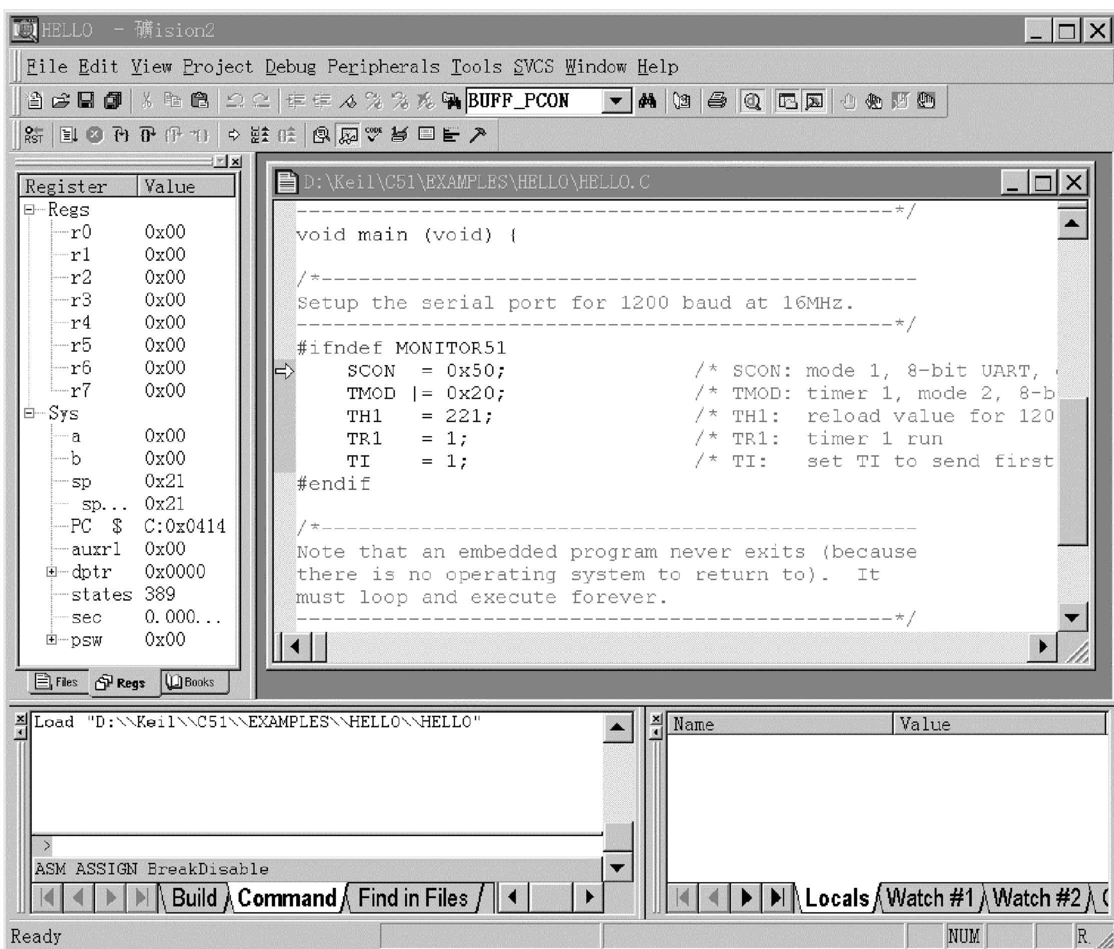
附图 13　Hello 工程的编辑/编译界面



附图 14　Hello 工程的编译结果输出

使用提供的示范程序不应该会有错误。

测试 HELLO：HELLO 程序被编译和链接后，您可以用 μVision2 调试器对它进行测试。在 μVision2 Debug 菜单或工具栏用 Start/Stop Debug Session 命令可以开始测试。μVision2 初始化调试器并启动运行程序直到 main 函数。显示的屏幕如附图 15 所示。



附图 15　Hello 工程的调试界面 1

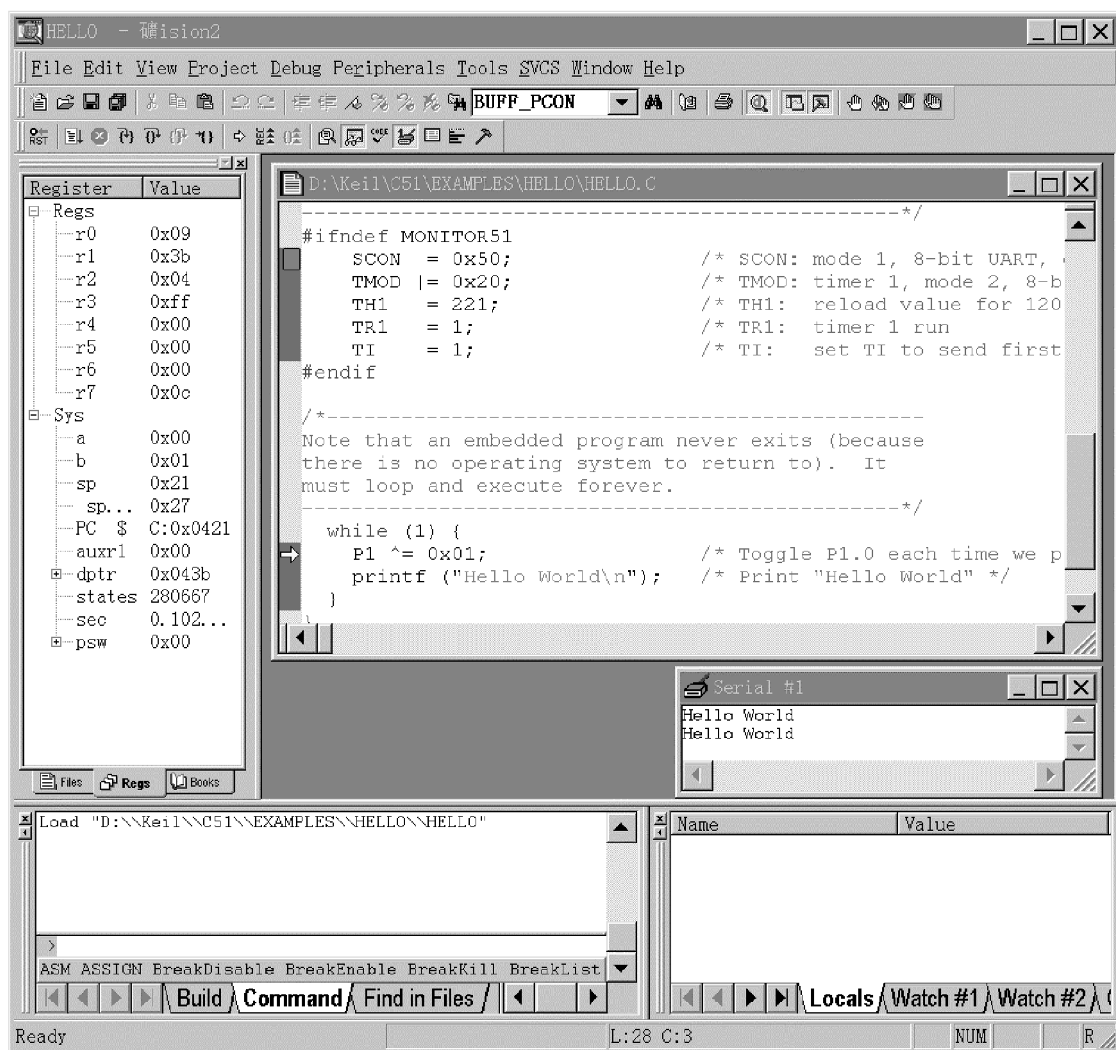
打开 Serial Window #1，用 View 菜单或 Debug 工具栏的 Serial Window #1 命令显示应用程序的串行 Serial 输出。

用 D ebug 菜单或工具栏的 Go 命令运行 Run HELLO 程序。HELLO 程序执行后在

Serial 窗口显示文字“Hello World”。在 HELLO 输出“Hello World”后，它开始执行一个无限的循环。

用 Debug 菜单或工具栏的 Halt 命令停止运行（Stop Running）HELLO 程序。您也可以在 Output 窗口的 Command 页输入 ESC 停止运行。

调试过程中 μVision2 会显示如附图 16 所示的输出。



附图 16　Hello 工程的调试界面 2

单步和断点：

用工具栏或鼠标右键打开 local 编辑器菜单的 Insert/Remove Breakpoints 命令，在 main 函数的开始处设置一个断点。

用 Debug 菜单或工具栏的 Reset CPU 命令。如果 HELLO 停止运行，Run 命令可以使

它继续执行，μVision2 会在断点处停止程序。

用 Debug 工具栏的 Step 按钮可以单步执行 HELLO 程序。当前的指令用黄色箭头标出。每执行一步箭头都会移动。

将鼠标移到一个变量上可以看到它们的值。

任何时间都可以用 Start/Stop Debug Session 命令停止调试。

参 考 文 献

［1］ 赵亮，侯国锐．单片机 C 语言编程与实例［M］．北京：人民邮电出版社，2003．

［2］ 求是科技．单片机典型模块设计实例导航．北京：人民邮电出版社，2004．

［3］ 张志良．单片机原理与控制技术［M］．北京：机械工业出版社，2001．

［4］ 胡同森，颜晖，董灵平，等．程序设计教程［M］．杭州：浙江科学技术出版社，2000．

［5］ 谭浩强，张基温，唐永炎．C 语言程序设计教程［M］．2 版．北京：高等教育出版社，1998．

［6］ 吴金戌，沈庆阳，郭庭吉．8051 单片机实践与应用［M］．北京：清华大学出版社，2002．

［7］ 傅扬烈．单片机原理与应用教程［M］．北京：电子工业出版社，2002．

［8］ 周兴华．手把手教你学单片机［M］．北京：北京航空航天大学出版社，2005．

［9］ 张毅刚，彭喜源，谭晓韵，等．MCS-51 单片机应用设计［M］．哈尔滨：哈尔滨工业大学出版社，2003．

［10］ 余永权．Flash 单片机原理及应用［M］．北京：电子工业出版社，1997．

［11］ I．Scott MacKenzie．THE 8051MICROCONTROLLER［M］．USA：Prentice-Hall．Inc．，1995．

［12］ 郝建国．家用电器遥控系统集成电路大全［M］．北京：人民邮电出版社，1996．

［13］ 谭浩强．C 程序设计［M］．北京：清华大学出版社，1991．

［14］ 马忠梅，籍顺心，张凯，等．单片机的 C 语言应用程序设计［M］．3 版．北京：北京航空航天大学出版社，2003．

［15］ 周坚．单片机轻松入门［M］．北京：北京航空航天大学出版社，2004．

ISBN 978-7-111-30335-0
ISBN 978-7-89451-488-2(光盘)
策划编辑：林春泉

地址：北京市百万庄大街22号
电话服务
社服务中心：(010)88361066
销售一部：(010)68326294
销售二部：(010)88379649
读者服务部：(010)68993821

邮政编码：100037
网络服务
门户网：http://www.cmpbook.com
教材网：http://www.cmpedu.com
封面无防伪标均为盗版

上架指导：工业技术／电子技术



定价：55.00元（含1CD）